Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

TRABAJO FIN DE GRADO

CloudRoom
Una Plataforma Universitaria Social, Semántica y Ubicua

Autor: Francisco Javier Sánchez Carmona

Directora: Sonia de Frutos Cid

Madrid, Enero de 2014

A Isabel, José María, José Ángel y Sonia.

Reconocimientos

No es fácil acordarse de todos aquellos que han pasado durante todos estos años por mi lado. Unos hace tiempo que abandonaron, otros se licenciaron, y muchos otros han ido desapareciendo sin hacer ruido poco a poco. Quiero sin embargo, agradecer a todos ellos la ayuda recibida durante mis años de carrera. Sin duda, todos y cada uno de ellos han aportado su granito de arena para que hoy esté aquí.

Me gustaría especialmente dedicar unas palabras a la persona que ha logrado en gran medida orientar mi rumbo en la universidad estos últimos años. Me refiero, por supuesto a mi tutora, Sonia. Ha conseguido hacerme ver el auténtico valor de lo que aquí se nos enseña, ayudándome a crecer académicamente. De cualquier otra forma, esto probablemente no hubiese sucedido. Este proyecto, CloudRoom, no existiría hoy sin su increíble apoyo durante el último año.

Por último, quiero agradecer a mi familia su esfuerzo incondicional, y su apoyo todos estos años, que ha sido determinante en la finalización de mis estudios.

Abstract

We are living through an age of Internetification. Nowadays, Internet connections are a utility whose presence one can simply assume. The web has become a place of generation of content by users. The information generated surpasses the notion with which the World Wide Web emerged because, in most cases, this content has been designed to be consumed by humans and not by machines. This fact implies a change of mindset in the way that we design systems; these systems should be able to support a computational and storage capacity that apparently grows endlessly. At the same time, our education system is in a state of crisis: the high costs of high-quality education threaten the academic world. With the use of technology, we could achieve an increase of productivity and quality, and a reduction of these costs in this field, which has remained largely unchanged since the Renaissance.

In CloudRoom, a MOOC platform has been designed with an architecture that satisfies the last conventions on Cloud Computing; which involves the use of REST services, NoSQL databases, and uses the last recommendations from W3C in terms of web development and Linked Data. For its building process, agile methods of Software Engineering, Human-Computer Interaction techniques, and state of the art technologies such as Neo4j, Redis, Node.js, AngularJS, Bootstrap, HTML5, CSS3 or Amazon Web Services have been used.

Furthermore, a comprehensive Informatics Engineering work has been performed, by combining virtually all of the areas of knowledge in Computer Science. Summarizing, the pillars of a robust, maintainable, and distributed system have been devised; a system with social and semantic capabilities, which runs in multiple devices, and scales to millions of users.

Extracto

Estamos viviendo la era de la Internetificación. A día de hoy, las conexiones a Internet se asumen presentes en nuestro entorno como una necesidad más. La Web, se ha convertido en un lugar de generación de contenido por los usuarios. Una información generada, que sobrepasa la idea con la que surgió esta, ya que en la mayoría de casos, su contenido no se ha diseñado más que para ser consumido por humanos, y no por máquinas. Esto supone un cambio de mentalidad en la forma en que diseñamos sistemas capaces de soportar una carga computacional y de almacenamiento que crece sin un fin aparente. Al mismo tiempo, vivimos un momento de crisis de la educación superior: los altos costes de una educación de calidad suponen una amenaza para el mundo académico. Mediante el uso de la tecnología, se puede lograr un incremento de la productividad, y una reducción en dichos costes en un campo, en el que apenas se ha avanzado desde el Renacimiento.

En CloudRoom se ha diseñado una plataforma MOOC con una arquitectura ajustada a las últimas convenciones en Cloud Computing, que implica el uso de Servicios REST, bases de datos NoSQL, y que hace uso de las últimas recomendaciones del W3C en materia de desarrollo web y Linked Data. Para su construcción, se ha hecho uso de métodos ágiles de Ingeniería del Software, técnicas de Interacción Persona-Ordenador, y tecnologías de última generación como Neo4j, Redis, Node.js, AngularJS, Bootstrap, HTML5, CSS3 o Amazon Web Services. Se ha realizado un trabajo integral de Ingeniería Informática, combinando prácticamente la totalidad de aquellas áreas de conocimiento fundamentales en Informática.

En definitiva se han ideado las bases de un sistema distribuido robusto, mantenible, con características sociales y semánticas, que puede ser ejecutado en múltiples dispositivos, y que es capaz de responder ante millones de usuarios.

Índice General

"La mente no es un vaso para llenar, sino una lámpara para encender"

Plutarco

Capítulo 1
Introducción

1.1 Motivación

Debemos asumirlo, actualmente estamos viviendo la era de la **Internetificación**[1], muy similar a la era de la Electrificación que tuvo lugar a principios del siglo XX. En unas décadas, la electricidad pasó de ser una simple curiosidad, a una necesidad mundial. Su interfaz de uso se ha convertido en algo tan sencillo como enchufarse a la red eléctrica. Prácticamente cualquier objeto imaginable se ha convertido en eléctrico. Algo similar está ocurriendo con Internet en el Siglo XXI: las conexiones a Internet han ido evolucionando desde los módems de 56k hasta las conexiones inalámbricas de banda ancha, de tal forma que estas últimas, hoy en día se las presupone presentes en nuestro entorno como una necesidad más. Estamos experimentando como Internet crece sin fin, y se extiende a casi cualquier dispositivo imaginable (Figura 1.1).

Figura 1.1 *20 años de diferencias*
(Fuente: Internet Trends por Mary Meeker, KPCB)

El número de usuarios de Internet en el año 2012[1] supera los 2400 millones de personas. Del variable ranking de las webs más visitadas del mundo, podemos encontrar entre las 10 primeras, a las principales **redes sociales**, como Facebook, Twitter o LinkedIn[3]. Desde la aparición del término Web 2.0[4] la web ya no es sólo un lugar de consulta para los usuarios, sino también de generación de contenido por los mismos. Esto supone un cambio de mentalidad no solo en la manera de ver la web, sino también en la forma en que diseñamos sistemas, que sean capaces de soportar una carga computacional y de almacenamiento que crece sin un fin aparente. Este aumento de la carga computacional y de almacenamiento se debe al auge de los sistemas distribuidos, potenciados a su vez debido al enorme desarrollo de las conexiones de acceso a Internet, y que han permitido acercar a casi cualquier persona del planeta una conexión barata y de calidad.

Una plataforma como Facebook, vista como un país, sería considerado como el tercero más poblado del planeta con 727 millones[5] de usuarios activos diariamente (Figura 1.2), tan sólo por detrás de China (1388 millones) e India (1257 millones)[6]. Este hecho transforma claramente la forma de hacer negocios.

Figura 1.2 *El mapa del Mundo de Facebook, Facebook Engineering*

Al tiempo que el número de usuarios, y la cantidad de contenido generado crece cada vez más deprisa, Internet está independizándose de los ordenadores. La **computación móvil** ya no es una tendencia, sino el futuro de la computación. Las aplicaciones

móviles aspiran a dejar atrás a las aplicaciones web de escritorio, al igual que las aplicaciones web dejaron atrás a las aplicaciones nativas de escritorio.

Con el surgimiento de las conexiones móviles de 4ª generación (4G), y con smartphones y tablets que cada vez tienen más capacidad de cómputo, el despegue de nuevos servicios para estos dispositivos no es sino cuestión de tiempo[7]. La computación móvil no es el fin de esta cadena; el denominado Internet de las cosas comienza a ser una realidad: Televisores, frigoríficos, coches, relojes y dispositivos vestibles, o las ciudades inteligentes, son sólo la punta del iceberg de lo que está por venir.

Toda esta información generada sobrepasa la idea con la que surgió la World Wide Web, ya que en la mayoría de los casos, el contenido de la Web no se ha diseñado más que para ser consumido por humanos, y no por máquinas, tal y como Tim Berners-Lee postuló en su día. El hecho es que en estos momentos, no tenemos una única Web, sino conjuntos de información desperdigada e inconexa a través de Internet. Las **tecnologías semánticas** serán vitales[8] para poder unificar la Web y aprovechar de manera óptima toda esa información generada.

En todo este contexto, entra en juego una nueva forma de entender y consumir la Informática, denominada **Cloud Computing**. Actualmente, nadie duda de que este paradigma de computación es el presente y el futuro de la computación distribuida. Se espera que pueda suponer un empuje vital al crecimiento económico mundial. El hecho de proveer recursos de TI bajo demanda a través de la red hace que las empresas y administraciones públicas encuentren en este modelo una forma de ahorrar en costes de uso y mantenimiento, así como también de aumentar su productividad.

En 2010, Vivek Kundra, asesor tecnológico de Barack Obama y el Jefe de Informática Federal de la Casa Blanca presentó ante el Congreso de los Estados Unidos al Cloud Computing como esencial y fundamental en la modernización de la infraestructura tecnológica del Gobierno[9].

En Septiembre de 2012, la Comisión europea, adoptó la estrategia “Liberar el potencial de la computación en nube en Europa” . Dicha estrategia pretende facilitar y potenciar la adopción de este paradigma tecnológico en todos los sectores de la economía con el objetivo de generar 2’5 millones de nuevos puestos de trabajo en Europa, así como también un incremento anual del PIB en la Unión igual a 160.000 millones de Euros, en torno a un 1% de aquí al 2020[10].

Por ejemplificar algunos usos cercanos al consumidor, podemos encontrar Gmail, Dropbox, o Facebook, pasando por Google Docs o Microsoft Office 365, auténticas soluciones Cloud de *Software as a Service.*

Sin embargo, si queremos ir más allá, encontramos usos de auténtico calado:

Figura 1.3 *Curiosity emite en directo desde Marte gracias a AWS, Adobe Blog*

En Agosto de 2012, la NASA logró con éxito aterrizar en Marte con su Misión Curiosity, un robot de exploración que mandó sus primeras imágenes a La Tierra (Figura 1.3), y fueron visibles en streaming gracias al uso de la Plataforma Cloud, Amazon Web Services[11].

Como hemos visto, estamos en un momento en el que Internet es un sitio increíblemente importante para cualquier tipo de actividad. El mundo de la Educación parecía aguardar latente su oportunidad, hasta ahora. En el 2012, las Universidades de Stanford, Harvard o el Instituto Tecnológico de Massachussets (MIT), comenzaron a impartir abiertamente cursos online en una modalidad hasta ahora inédita; gracias a sus plataformas web, crearon un modelo masivo, abierto y gratuito, ofrecido a través

de Internet para todo el mundo; este modelo se ha denominado **Massive Open Online Courses**, en adelante **MOOC**. El éxito de estas plataformas de cursos online ha sido fulgurante[12]. El primer curso de "Introducción a la Inteligencia Artificial" de Sebastian Thrun y Michael Sokolsky propició una avalancha de más de 150.000 suscriptores en su plataforma *Udacity* . *edX*, la plataforma de Harvard y el MIT, acoge aproximadamente 1 millón de usuarios registrados. Nada comparado con *Coursera* que tiene actualmente 4 millones de estudiantes enrolados en cursos por todo el mundo, y con una celeridad en su crecimiento que supera a la que tuvo Facebook en el momento de su nacimiento. El caso de *Coursera* es el más impresionante de todos. Su creación surgió de la mente dos profesores de Ingeniería Informática de la Universidad de Stanford, *Daphne Koller* y *Andrew Ng*, que observaron que sus clases en Youtube, tenían más visitas que las mismas presenciales.

Para poner de manifiesto el poder que tienen este tipo de plataformas, Andrew Ng pone un ejemplo particular. Un curso tradicional de la Universidad de Stanford tiene 400 estudiantes de media durante el curso. En 2011, un MOOC que se realizó de ese mismo curso en Stanford, alcanzó una audiencia de 100.000 estudiantes. Para poner este dato en contexto, enseñar de manera tradicional a un número equiparable de estudiantes le llevaría a un docente 250 años.

Daphne Koller, afirma[13] que vivimos un momento de crisis de la educación superior, tanto en Estados Unidos como en el resto del Mundo. Los costes de una educación superior de calidad suponen una amenaza para el mundo académico, y debemos reducir dichos costes, al mismo tiempo que mejoramos su calidad, afirma Koller. Asímismo, sugiere que aunque parezca contradictorio, esto supone un beneficio, fácilmente demostrable con datos históricos: "En el siglo XIX, el 60% de la fuerza trabajadora de EE.UU se concentraba en la agricultura, y había frecuente escasez de alimentos. Hoy en día, la agricultura sólo supone el 2% de la fuerza trabajadora, y hay excedentes de alimentos". La razón de esto, según Koller, es el uso de la tecnología, que ha supuesto un incremento de la productividad. En la educación sin embargo, apenas se ha avanzado desde el Renacimiento.

1.2 Objetivos

En un primer momento, los objetivos de este trabajo eran los de construir una red social colaborativa, con algunos elementos educativos. La idea fue evolucionando hasta

convertirse en una plataforma de MOOCs llamada **CloudRoom**. El sistema además debía ser diseñado pensando en soportar la carga de millones de usuarios.

Esta plataforma es fundamentalmente un Sistema distribuido, cuya arquitectura debía ajustarse a las últimas convenciones en diseño de sistemas en el campo del Cloud Computing. Utilizando el estilo arquitectural RESTful, para el diseño de cada uno de los servicios de la plataforma, y de bases de datos NoSQL. Haciendo uso de las últimas recomendaciones del W3C, como HTML5, y prestando atención a OWL y RDF.

Este documento se estructura en cinco capítulos. En el primero de ellos se abordará el estado actual de la cuestión. Después, en el segundo, se describirá el problema a tratar en este trabajo, y se continuará en el capítulo 3 con la solución propuesta para el mismo. Las conclusiones extraídas de este proyecto se comentan en el cuarto capítulo, y aquellos apartados que podrían desarrollarse en un futuro si el proyecto tuviera continuidad, quedan detalladas en el quinto y último capítulo.

Capítulo 2
Estado de la Cuestión

A continuación, se estudiarán las características de los MOOCs, así como sus metodologías de aprendizaje y evaluación. Por otro lado, se tratará la última situación tecnológica en Cloud Computing y otras tecnologías relacionadas con el propósito de este Trabajo, como son las bases de datos NoSQL, prestando especial atención a las bases de datos orientadas a grafos, las tecnologías semánticas centradas en datos enlazados y la Web Semántica, y como las grandes cantidades generadas por las Redes Sociales afectan a cómo debe afrontarse la gestión de la masiva cantidad de información que estas generan. Por último, se dedicará una parte final para analizar el panorama de la computación móvil actual y futuro, y los principales actores en este campo.

El objetivo de este capítulo es el de estudiar todo aquello relacionado con el tema de este trabajo, así como también estudiar y documentar el estado de la técnica de las tecnologías utilizadas en el contexto de los sistemas como el que se pretende construir. Se repasaran tecnologías utilizadas y se detallaran más aquellas que se han utilizado en el desarrollo de este proyecto.

2.1 El Año de los MOOCs

El año 2012 fue el año de irrupción de esta tendencia en la educación online. 2013 ha sido definido por el diario *The New York Times* como: "*The Year of the MOOCs*" acorde con lo que publicaba este diario sobre este fenómeno en auge. Sin embargo, según Gartner 2014 supone un descenso en su interés, y 2015 supondrá un punto de inflexión hacia el crecimiento lento pero progresivo de esta tendencia (Figura 2.1).

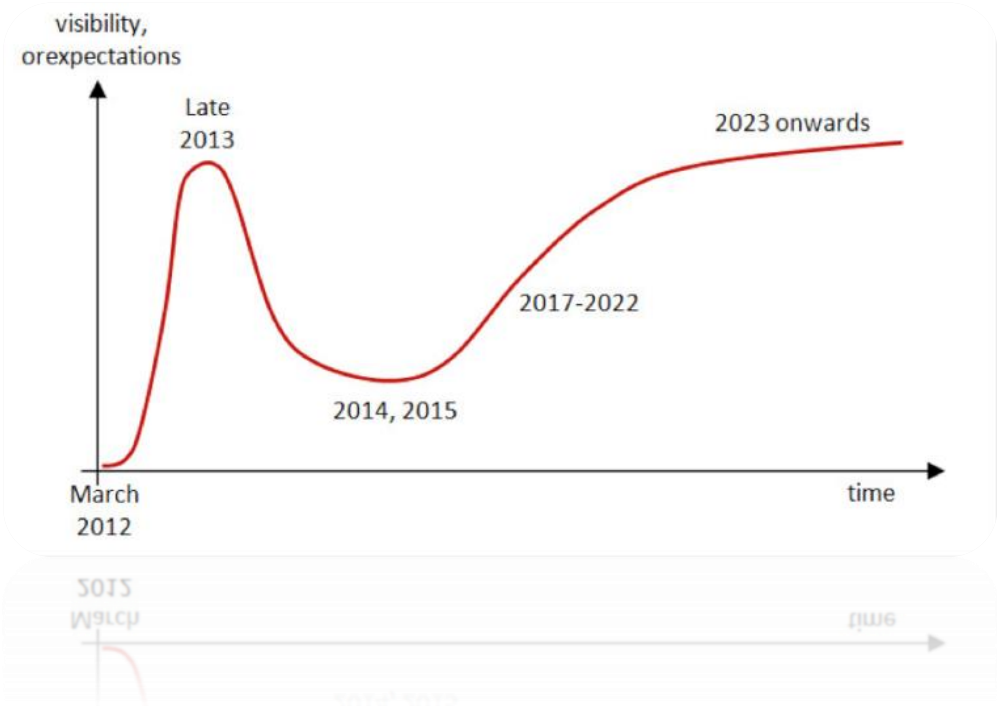

Figura 2.1 *Gartner Hype Cycle: MOOCs (Fuente: Pandodaily)*

Este tipo de cursos aprovechan la infraestructura de Internet para impartirse, y se caracterizan por ser de generalmente gratuitos, abiertos a todo el mundo, y consecuentemente ofrecidos a una audiencia masiva.

El primer MOOC tuvo su origen en un experimento de la Universidad de Stanford, pero su éxito ha propiciado que hayan emergido otras plataformas que ofrecen el mismo servicio. Las más importantes en este momento son las que siguen a continuación.

Coursera*:* Creada por profesores de Stanford, esta plataforma se ha ido asociando con las mejores Universidades del Mundo para ofrecer cursos personalizados de Medicina, Biología, Humanidades, Ciencias Sociales, Negocio, Informática y un largo rango de temas. Es el líder actual, con 4 millones de suscriptores (Figura 2.2). Ha recibido importantes inyecciones de capital riesgo, y en 2012 fue nombrada mejor Startup por algunos medios tecnológicos[14].

Hasta hace poco era completamente gratuita, ahora ofrece la posibilidad de obtener certificados firmados oficialmente por algunas Instituciones que cobran solamente si el

alumno ha aprobado con un margen determinado, y solo si este quiere dicho certificado.

Está en pleno desarrollo, y tecnológicamente avanza constantemente, ahora introduciendo búsqueda semántica para encontrar cursos. Además, hace una apuesta por nuevas metodologías educativas a través de preguntas o tests intercaladas en los videos de las clases.

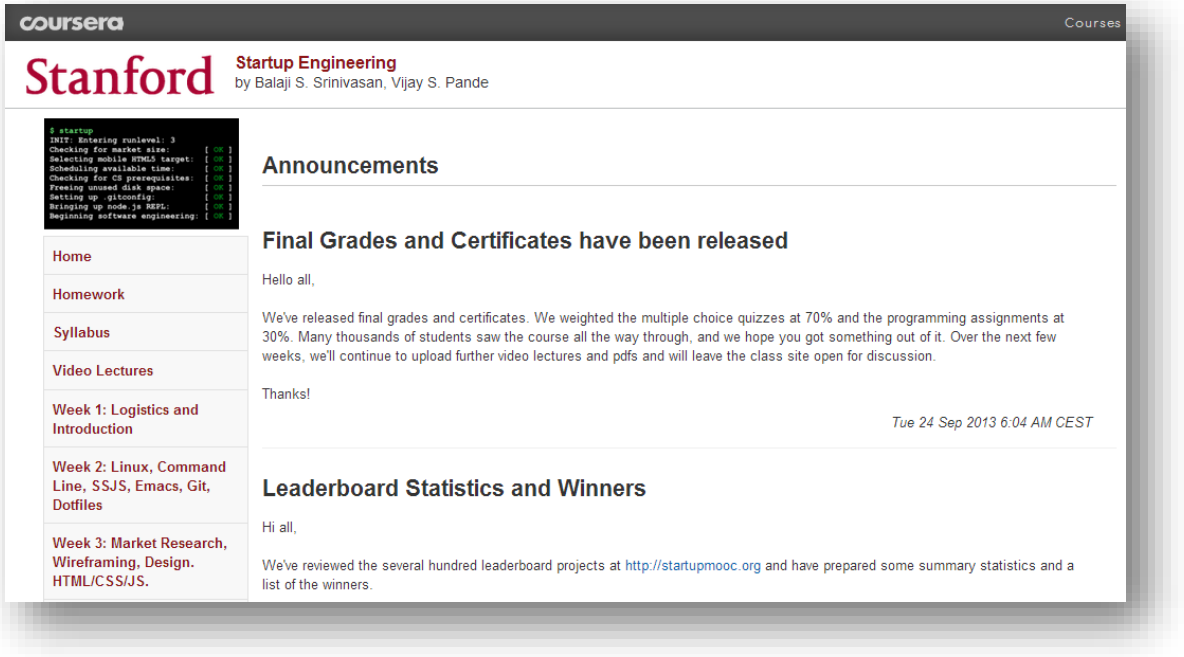

Figura 2.2 *Interfaz de usuario de Coursera*

edX: De la misma manera que Coursera, edX es una plataforma que se asocia con las mejores Universidades del Mundo para ofrecer cursos. Su funcionamiento es muy similar, y también está aplicando políticas para obtener beneficios en función de certificados para los cursos (Figura 2.3). Sin embargo su alcance no es el de Coursera. Su reciente asociación con Google, le ha fortificado, que usará su plataforma open source para construir la suya propia, mooc.org.

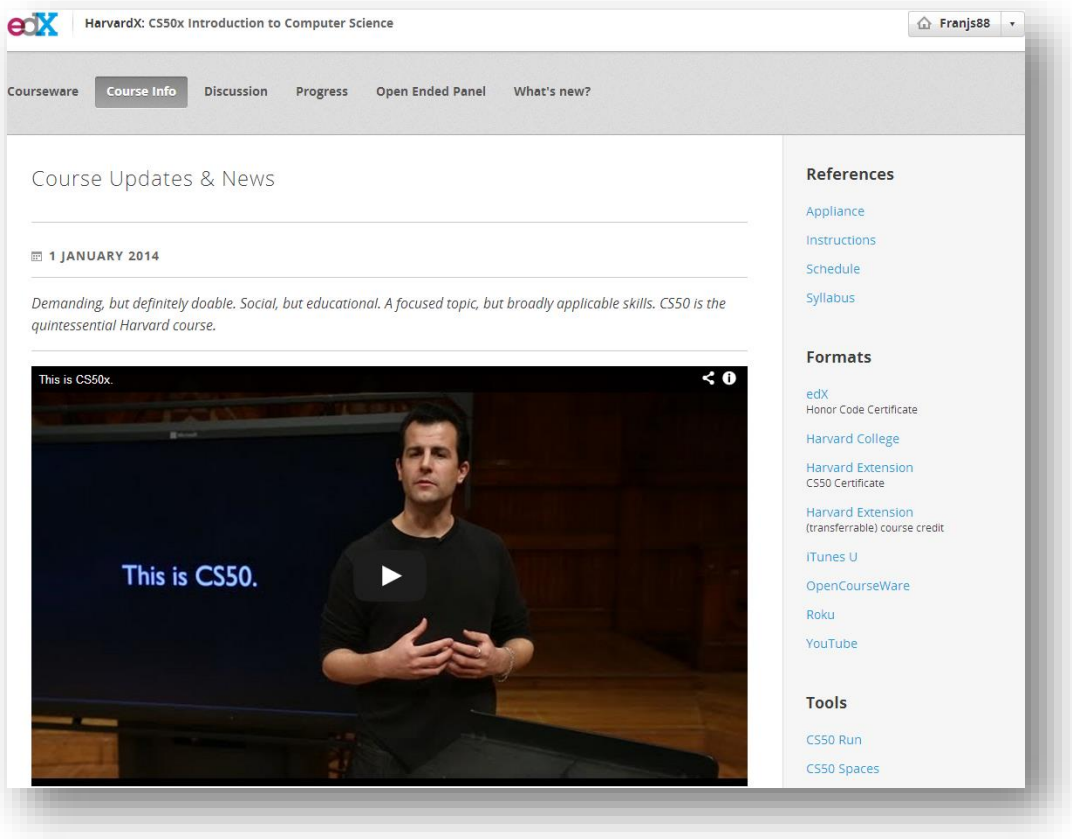

Figura 2.3 *Interfaz de usuario de edX*

Udacity: Creada también en la Universidad de Stanford, pero por diferentes personas, surgió al contrario que Coursera, de manera accidental. A diferencia de otras plataformas, ofrece cursos de empresas importantes que cobran por dichos cursos, como Facebook, o MongoDB, que dan cursos relacionados con las actividades de su organización. La enseñanza por tanto no es gratuita en todos sus casos, algo que va un tanto en contra de la filosofía MOOC (Figura 2.4).

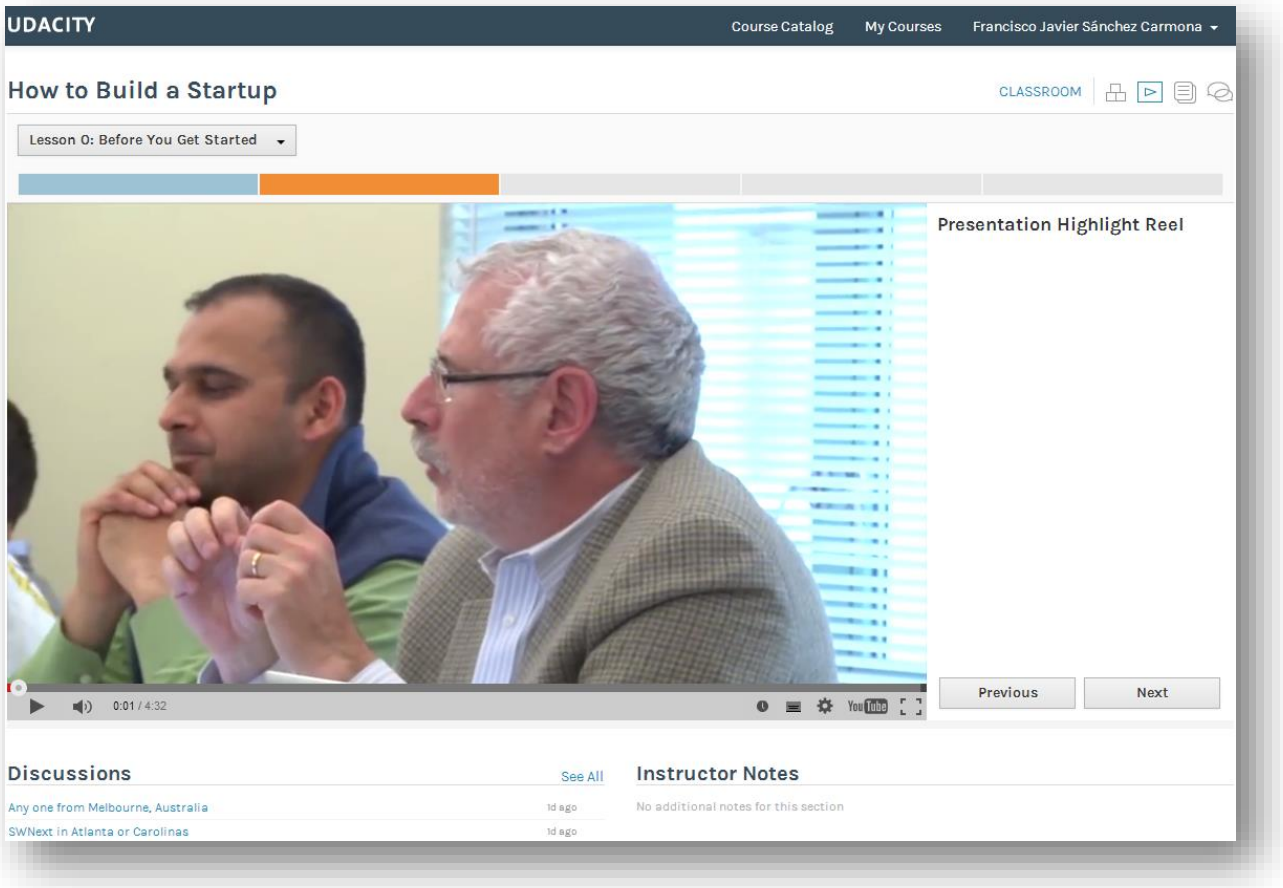

Figura 2.4 *Interfaz de usuario de Udacity*

Khan Academy: Apoyada por la fundación Bill & Melinda Gates, entre otras, surgió como una iniciativa sin ánimo de lucro para llevar la educación a cualquier persona (Figura 2.5). Fue creado por Salman Khan, un educador graduado en el MIT y Harvard.

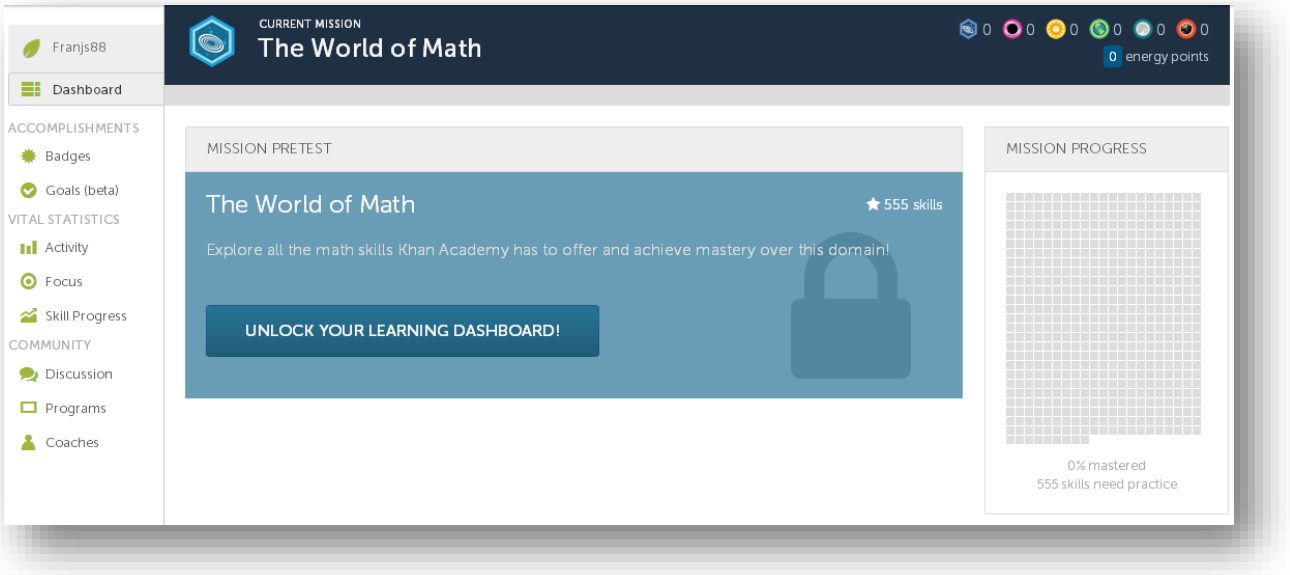

Figura 2.5 *Interfaz de usuario de Khan Academy*

mooc.org: Muy poco se sabe de esta plataforma aún en desarrollo; salvo que Google está detrás de ella (Figura 2.6). Tras un acuerdo con edX, utilizará parte de su infraestructura Open-Source para gestionarse. Su lanzamiento está previsto para 2014.

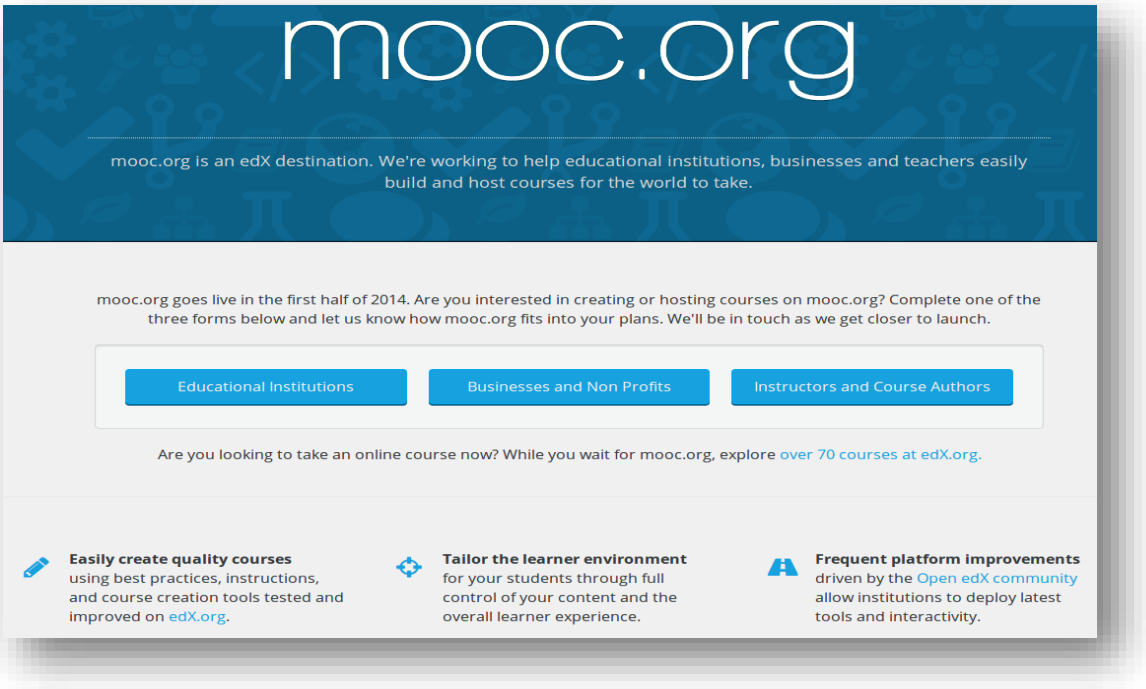

Figura 2.6 *Sitio web oficial de mooc.org*

MiriadaX: Plataforma líder en España. Creada entre Banco Santander y Telefónica. Oferta MOOCs de numerosas Universidades Españolas. Se orienta al mercado de habla hispana y portuguesa. Presenta muchas de las funcionalidades educativas que las anteriores plataformas, sin embargo, tecnológicamente está varios escalones por detrás. Su punto fuerte es el gran número de cursos en castellano, y de universidades españolas de que dispone (Figura 2.7).

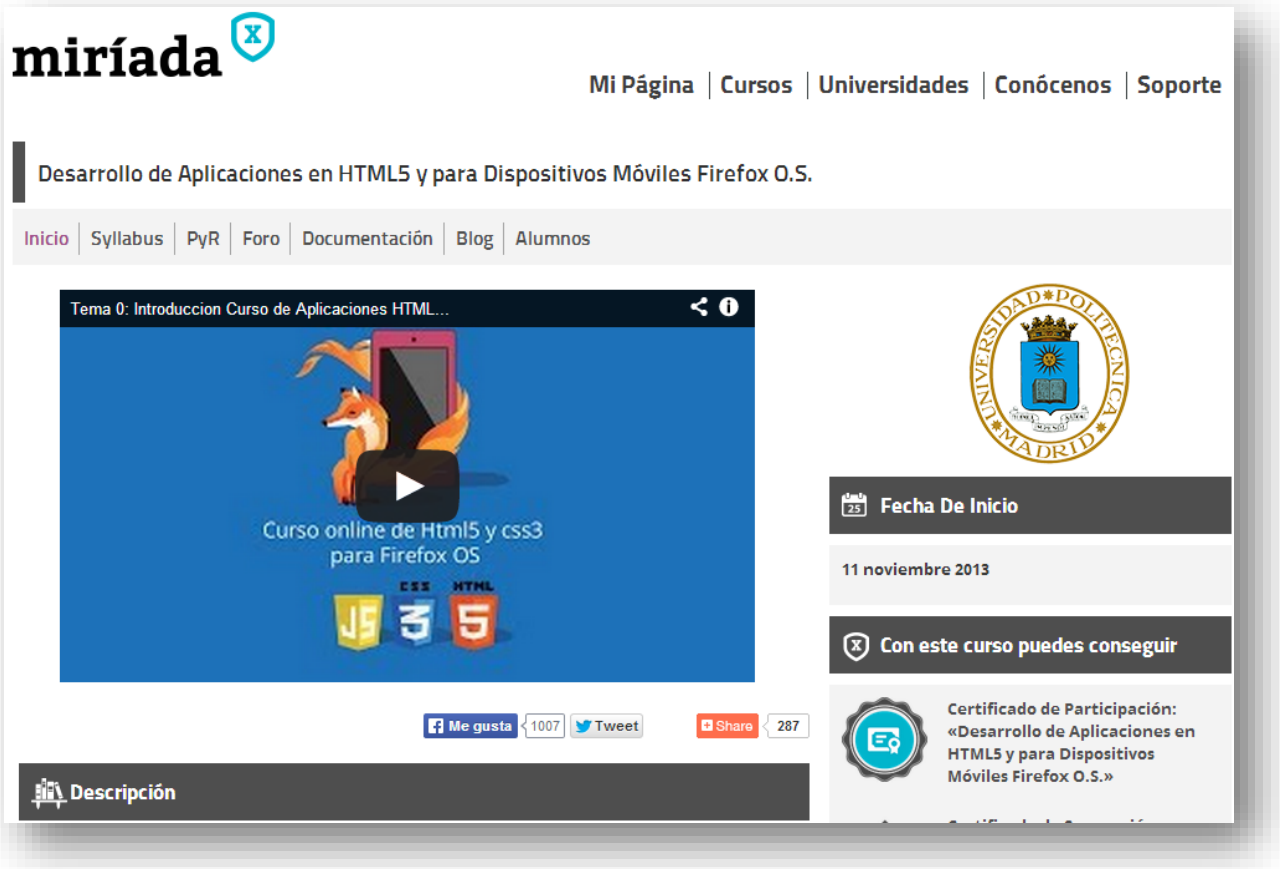

Figura 2.7 *Interfaz de usuario de MiriadaX*

Al contrario que los *OpenCourseWare*, en los que las Universidades ofrecen abiertamente sus materiales de sus cursos como si se pudiera asistir de oyente a sus clases, los MOOCs presentan cursos personalizados, orientados directamente al estudiante, con metodologías especiales, en las que se pretende que los contenidos se sigan en una franja temporal de días, pero siempre dejando que cada alumno se organice el tiempo en el que consume los materiales de la forma que desee.

La forma de corregir proyectos, tests o recursos para evaluar al estudiante, es también diferente:

- Los tests son una de las opciones clásicas.

- Evaluación entre iguales (P2P), en la que los alumnos se corrigen entre ellos.
- Coursera implementa un sistema de preguntas intercaladas entre los videos de las lecciones, de forma que cuando un profesor quiera hacer una pregunta a los alumnos, todos y cada uno de ellos puedan responder e interactuar con la materia de una forma práctica.
- Existen formas de evaluar prácticas y proyectos de programación

Es necesario mejorar la forma de corregir preguntas de respuesta corta, o de desarrollo, que son importantísimas en materias de humanidades y letras, pero que al tener un carácter masivo, se hace imposible corregir individualmente a los alumnos, todo debe estar automatizado.

Los MOOCs presentan algunas funcionalidades sociales, sin embargo, son muy reducidas: La forma en que el profesorado puede interaccionar de manera más directa suele ser mediante la creación de Foros, en los que además los alumnos pueden colaborar entre ellos, y el profesor puede tener un feedback por parte de los estudiantes. A continuación se enumeran algunas de las diferencias y carencias con respecto a las redes sociales puras:

- Interacción entre usuarios y profesores en Foros tradicionales.
- Ausencia de talleres o herramientas colaborativas.
- Colaboración entre usuarios se limita a foros y en el caso de Coursera, a un sistema de quedadas en ciudades, al estilo de comunidades.
- Ausencia de Chats o canales IRC de comunicación instantánea.
- Imposibilidad de crear relaciones sociales entre alumnos con el fin de colaborar.
- Imposibilidad de crear grupos.

Actualmente la mayor parte de las plataformas de MOOCs son gratuitas, sin embargo, muchas están empezando a buscar modelos de negocio que les permitan pagar a sus inversores. Uno de los modelos que más visos tiene de implantarse, es el del pago por certificaciones, en el que una vez completado y aprobado un curso, el alumno, si lo desea, puede obtener una certificación por parte de la institución que lo imparta, a cambio de una suma de dinero.

2.2 Nacimiento y Auge del Cloud Computing

En medio de todo este contexto, se ha recuperado un concepto que John McCarthy[15] sugirió por primera vez en los años 60. La denominada *Utility Computing*, la idea de la Informática como suministro, como el agua o la electricidad. Una idea muy popular durante los años 60 que se fue apagando a mitad de los 90, pero que desde el año 2000, ha resurgido con fuerza[16].

> *"If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility... The computer utility could become the basis of a new and important industry."*
>
> *—John McCarthy, discurso en el centenario del MIT en 1961*

Las palabras de McCarthy (quien acuñó por primera vez el término *Inteligencia Artificial)*, no se perdieron en el tiempo. Años más tarde, Douglas Parkhill, en su libro *The Challenge of the Computer Utility*, exploró a fondo muchas de las capacidades actuales del Cloud Computing moderno, tales como el aprovisionamiento elástico de servicios.

Sentadas las bases teóricas, en los años 90 los proveedores de telecomunicaciones empezaron a ofrecer servicios de Internet a los usuarios de a pie con una buena calidad de servicio. Se introdujo el símbolo de una nube (Figura 2.8) para representar tanto las redes de las que era responsable el operador, como aquellas de las que lo eran los usuarios.

Quién acuñó el término de *Cloud Computing* es algo incierto[17]. Algunos creen que su primer uso fue el 9 de Agosto de 2006, por parte del CEO de Google, Eric Schmidt[18] durante una conferencia:

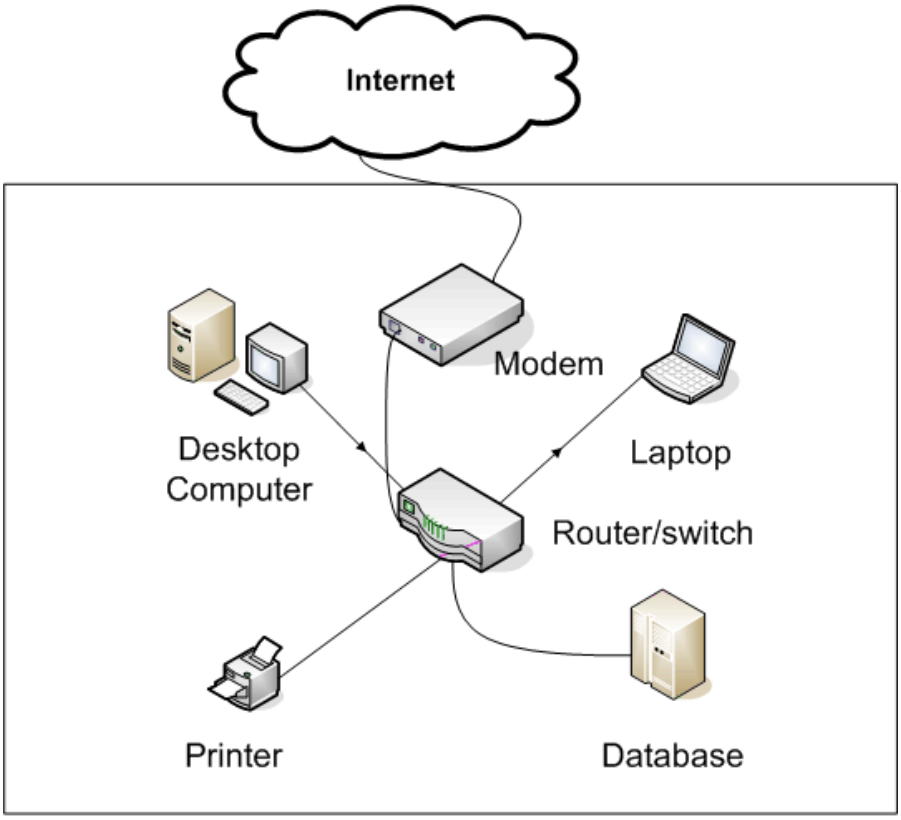

Figura 2.8 *La nube de Internet*

> *"I don't think people have really understood how big this opportunity really is. It starts with the premise that the data services and architecture should be on servers. We call it cloud computing – they should be in a "cloud" somewhere. And that if you have the right kind of browser or the right kind of access, it doesn't matter whether you have a PC or a Mac or a mobile phone or a BlackBerry or what have you – or new devices still to be developed – you can get access to the cloud."*
>
> —Eric Schmidt, "Search Engine Strategies Conference", 2006

El año siguiente el término empezó a usarse más ampliamente cuando compañías como Amazon, Microsoft o IBM empezaron a ofrecer sus propios servicios Cloud.

Lo que ha cambiado con respecto a lo McCarthy postuló en su día, se conoce como **Virtualización** (Figura 2.9). Los avances en Sistemas Operativos han permitido tener

un computador de ocho núcleos, y simular ocho computadores independientes, cada uno de ellos pudiendo ser formateado y reiniciado sin afectar a los otros siete.

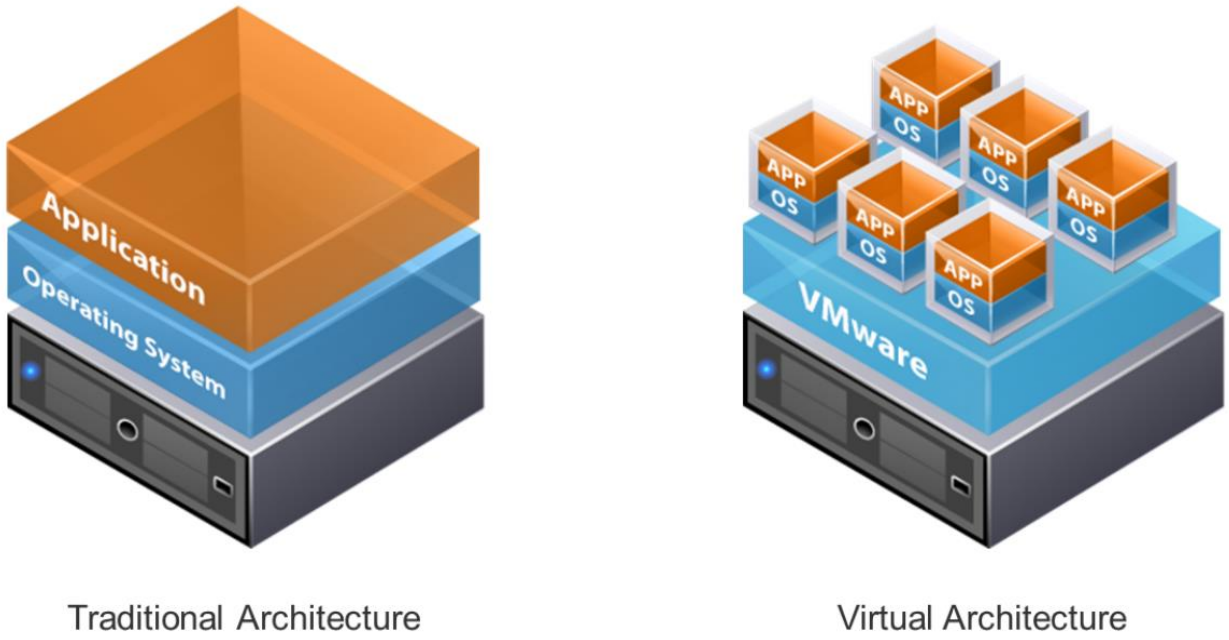

Figura 2.9 *El concepto de Virtualización ilustrado por VMware*

Esto permite a las empresas proveedoras de servicios Cloud, ofrecer un pago por servicio, escalable según la necesidad, ya que los proveedores Cloud pueden maximizar el uso de Hardware extremadamente caro y amortizar sus costes.

Aunque el rendimiento ofrecido por los computadores virtualizados es considerablemente bueno, no es el mismo que el de un computador físico. Además, la virtualización, no es la panacea, y una excesiva carga de trabajo en uno de los clientes de la misma máquina física puede afectar seriamente al rendimiento de los demás.

Existen diversos tipos de nubes en referencia al tipo de privacidad que se ofrece el servicio:

- **Cloud Pública**: Es el concepto tradicional conocido de forma general. El servicio se ofrece al público y está gestionado por un proveedor Cloud.
- **Cloud Privada**: Se crea un nube dentro de una red privada, que es accedida tan solo por unos pocos, y no está abierta al público.
- **Cloud Híbrida**: Se combinan dos o más nubes públicas y privadas. Una vez alcanzado el límite en una privada se puede acceder a los recursos ofrecidos por la pública.

- **Cloud Comunitaria**: Una infraestructura Cloud compartida por varias organizaciones y que da servicio a una comunidad. Puede ser gestionada por dichas organizaciones o por una tercera ajena.

Los proveedores de servicios Cloud ofrecen estos en distintos niveles y modelos de servicio fundamentales (Figura 2.10):

- **Infrastructure as a Service (IaaS):** Es el nivel básico. El usuario administra, controla y usa su sistema operativo y aplicaciones en los servidores del proveedor, y puede gestionar otros recursos como virtualización u otros sistemas operativos. Algunos de los proveedores más importantes de IaaS son Amazon, IBM y Oracle.
- **Platform as a Service (PaaS):** Nivel intermedio. El cliente ejecuta sus aplicaciones en la plataforma del proveedor con su sistema operativo y herramientas. En este nivel encontramos proveedores como Google App Engine, Heroku, o Windows Azure.
- **Software as a Service (SaaS):** Es el nivel que el usuario tradicional percibe. Aplicaciones web del proveedor de la aplicación, que se acceden como un servicio a través de la plataforma del proveedor Cloud.
- **XaaS**: Nuevos términos se han definido para distintos modelos, como *Database as a Service* (DaaS), *Network as a Service* (NaaS) y muchos otros.

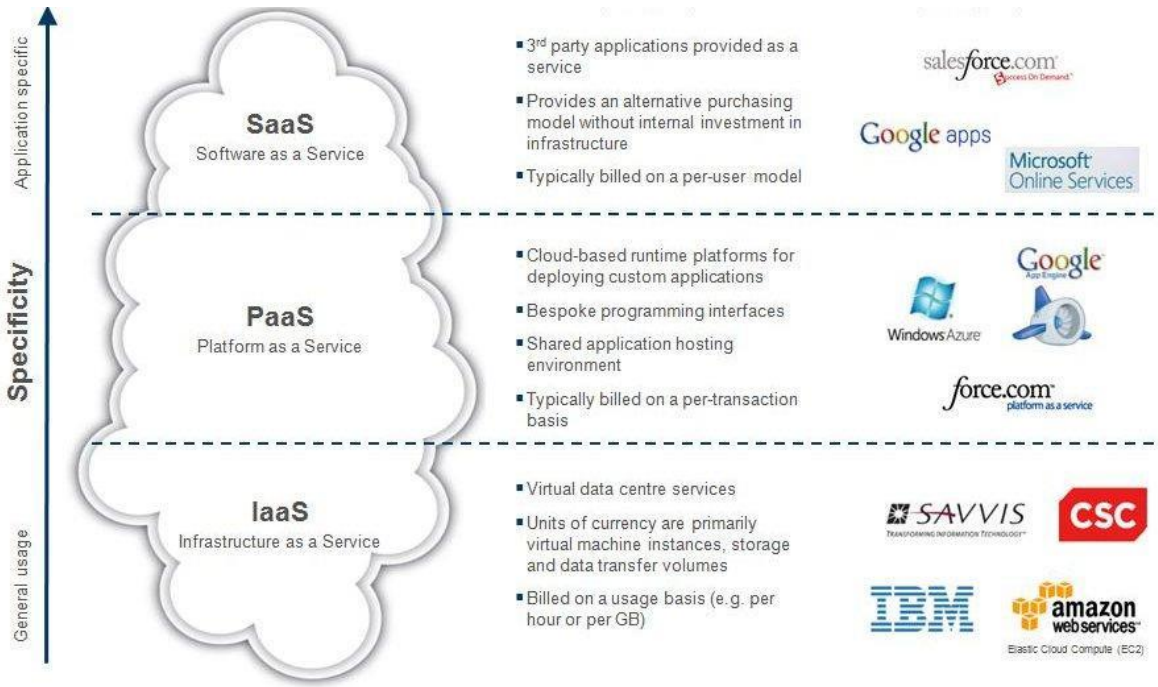

Figura 2.10 *Arquitectura de Capas del Cloud Computing según citihub*

Dentro de cada uno de estos niveles podemos encontrar diferentes proveedores, pero quizá el más importante de todos ellos sea Amazon, que ofrece una amplia variedad de servicios en su plataforma *Amazon Web Services*.

AWS[19] es una plataforma IaaS que ofrece decenas de servicios bajo demanda. Los más destacados son:

- **Elastic Compute Cloud (EC2):** ofrece capacidad informática en la nube de manera escalable, en forma de servidores configurables desde una fracción mínima de uso hasta lo que se necesite, de manera rápida.
- **AutoScaling:** Permite escalar EC2 de forma automática según su uso.
- **Elastic Load Balancing:** Permite redireccionar solicitudes entrantes a distintas instancias de EC2 de manera automática.
- **Elastic Block Store (EBS):** Permite utilizar almacenamiento en bloques que puede asociarse a una instancia y persiste a la vida de la misma.
- **Simple Storage Service (S3):** es un servicio de almacenamiento escalable para desarrolladores, permite a través de una interfaz web acceso a Buckets de una capacidad de hasta 5GB. Se pueden adquirir más Buckets en función de su necesidad.
- **CloudFront:** Es un servicio de Red de Distribución de Contenido (CDN) para desarrolladores, permite ofrecer contenido a baja latencia con independencia de la localización geográfica del cliente. Esto se consigue mediante una red global que en función de la localización del cliente, ofrece el contenido, tanto estático, como dinámico o de cualquier tipo, desde una localización óptima para obtener el mejor rendimiento posible.
- **Simple Notification Service (SNS):** Servicio de notificaciones push para dispositivos móviles, como iPhone, Android y similares, sencillo y automatizado. Permite además envío de SMS, y Email, además envíos a colas de mensajes, cualquier destino HTTP.
- **Simple Email Service (SES):** Servicio para envío masivo y transaccional de correo electrónico económico y muy ampliable para desarrolladores y empresas.
- **CloudWatch:** Servicio para monitorización de recursos y aplicaciones en la nube de Amazon. Permite gestionar alarmas sobre los recursos consumidos por el usuario.
- **Virtual Private Cloud:** Permite crear una red virtual dentro de la nube de Amazon, y controlar todos los aspectos del entorno de red virtual, incluido el

direccionamiento IP, la creación de subredes y configuración de tablas de rutas y puertas de enlace de red.

Dropbox, Instagram o Pinterest utilizan AWS. Spotify almacena toda su música en S3. Actualmente Amazon es el líder en aprovisionamiento Cloud.

Además de sus numerosos clientes que utilizan AWS directamente para desplegar sus aplicaciones o gestionar sus servicios, otras plataformas PaaS lo utilizan de manera subyacente para proporcionar un servicio de más alto nivel.

La adopción del Cloud Computing es cada vez mayor, sin embargo presenta serias barreras debido a los problemas de seguridad asociados a los datos. Los clientes presentan dudas sobre la privacidad de sus datos y la disponibilidad de los mismos.

Richard Stallman, conocido por su relación con el movimiento del software libre, manifestaba[20] que este paradigma presenta una trampa para los usuarios.

> *"It's stupidity. It's worse than stupidity: it's a marketing hype campaign. Somebody is saying this is inevitable — and whenever you hear somebody saying that, it's very likely to be a set of businesses campaigning to make it true."*
>
> Richard Stallman, citado por The Guardian, 29 de Septiembre, 2008

La disponibilidad del servicio es otro de los puntos negativos a los que se enfrenta; las aplicaciones SaaS han creado una sensación de disponibilidad total. Si cualquier usuario fuera a Google a buscar algo y estuviera sin servicio, pensarían que Internet se ha venido abajo. Por otro lado, este nuevo paradigma requiere de muchas herramientas nuevas que para muchas organizaciones supone un coste debido a la obligatoriedad de actualizar sus sistemas ya funcionales.

2.3 El Fenómeno NoSQL

Para poder construir sistemas que soporten una cantidad de usuarios masiva se hace necesario utilizar tecnologías que no han sido tradicionalmente adoptadas en los sistemas informáticos. En los últimos años, la popularidad de un tipo de tecnologías de almacenamiento conocido como NoSQL[21] (acrónimo de *Not Only SQL*), ha sufrido

un crecimiento asombroso. Así, el más importante ejemplo de esto es el gigante de Internet, Google, que ostenta desde hace años el primer puesto en número de visitas en Internet.

Para organizar la información de sus búsquedas y poder dar una respuesta rápida a esos millones de usuarios, Google hace uso de este tipo de bases de datos.

NoSQL es un término pobremente definido en informática actualmente: Se dice de ellas que poseen una especial facilidad para fragmentarse y utilizarse en entornos clúster (lo que les hace responder bien en entornos Cloud), suelen estar desprovistas de esquema, y escalan de manera sencilla.

La facilidad de fragmentación es uno de los aspectos más importantes por los que se usan, pero no el único. En ocasiones, este tipo de bases de datos encajan mejor en un determinado contexto de aplicación.

Muchas de las aplicaciones web en el mercado podrían ejecutarse en una base de datos relacional. Sin embargo, la complejidad de los datos ha alcanzado en algunos escenarios, cotas que eran inimaginables años atrás. Estos niveles de complejidad son: un volumen de datos que crece exponencialmente, un auge de las interrelaciones entre los datos y un esquema conceptual que varía gradualmente.

Pero no es solo el volumen de datos el causante del auge de este tipo de tecnologías. La velocidad a la que se generan datos, las relaciones complejas entre los mismos, o la desestructuración completa de la información, desborda la capacidad de los SGBD relacionales. Concretamente, este tipo de bases de datos difieren con las bases de datos relacionales en los siguientes puntos[22]:

- No usan SQL como lenguaje principal de consulta
- No requieren estructuras fijas como tablas para almacenar los datos
- No suelen garantizar transacciones
- No suelen soportar operaciones JOIN
- Suelen facilitar su integración en arquitecturas distribuidas
- Suelen presentar escalabilidad horizontal

Así podemos distinguir cuatro tipos principales (Figura 2.11):

- **Almacenes Clave-Valor:** Las más simples de todas; están formadas por un nombre de atributo (clave) y su valor. Algunas bases de datos clave-valor permiten añadir a cada valor un tipo, como es el caso de Redis.
- **Columnares:** Como Cassandra o HBase están optimizadas para consultas en grandes volúmenes de datos, y almacenan la información en columnas en lugar de filas.
- **Orientadas a Documentos:** Asocian cada clave con una estructura de datos compleja denominada documento. Los documentos pueden contener diferentes pares clave-valor, o clave-array, e incluso documentos anidados en colecciones. MongoDB o CouchDB.
- **Orientadas a Grafos:** Utilizadas sobre todo para almacenar información de redes; como las conexiones sociales. Neo4j y HyperGraphDB, son dos de las más conocidas.

Este tipo de bases de datos se alejan del modelo relacional postulado por Codd[23] y se distinguen por una mayor flexibilidad a costa de una pérdida en la integridad de los datos.

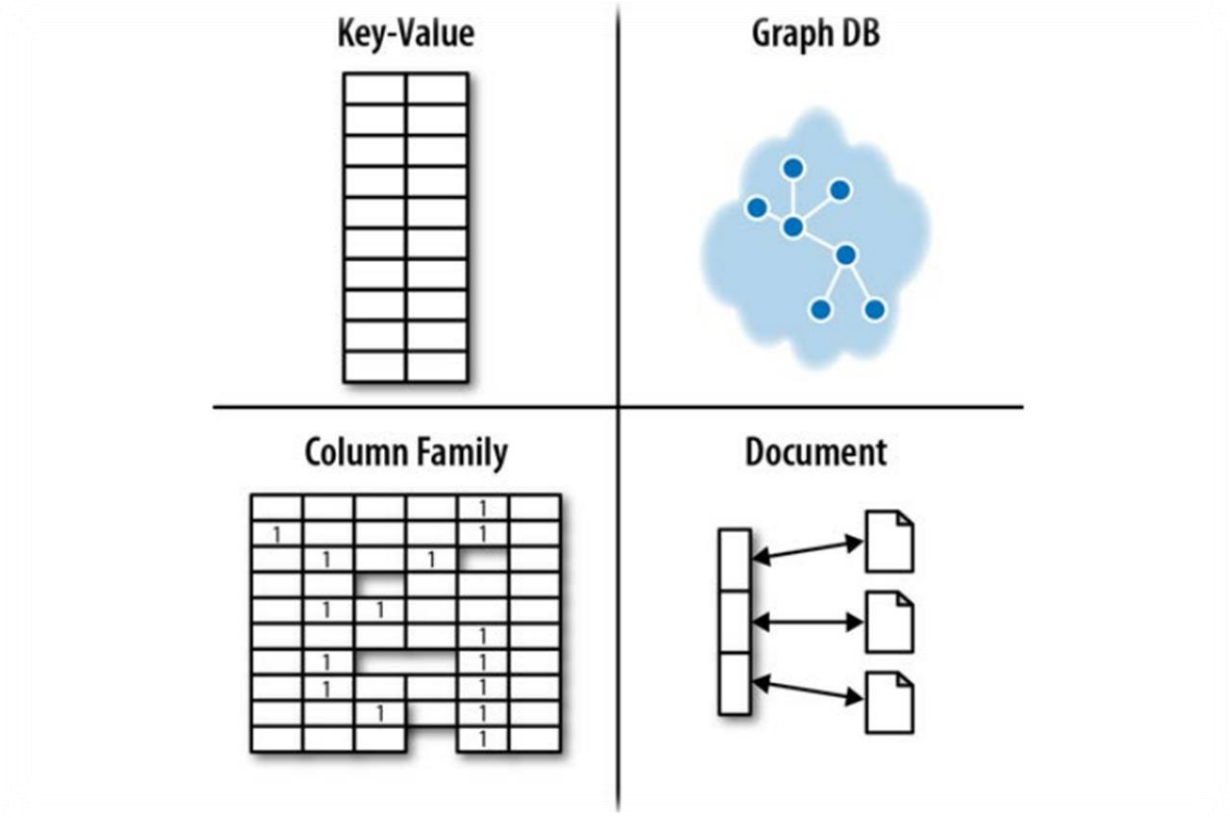

Figura 2.11 *Principales tipos de Bases de datos NoSQL (Fuente: Graph Databases)*

No solo el modelo de datos y el de consulta es diferente, el modelo de consistencia también lo es. En el mundo de las bases de datos relacionales estamos acostumbrados a las transacciones *ACID*, aquellas que garantizan un entorno fiable sobre el que operar con los datos:

Atomic
: Asegura que todas las operaciones en una transacción se han realizado con éxito, si se produce un fallo en mitad de una transacción todas las operaciones se revierten.

Consistent
: Garantiza la integridad de los datos. Esta propiedad sostiene que cualquier operación llevará a la base de datos de un estado válido a otro también valido.

Isolated
: Esta propiedad asegura que no una transacción no afectará a otras.

Durable
: Los resultados de una transacción son permanentes.

En muchos casos de uso, este modelo de consistencia es demasiado pesimista. En el mundo NoSQL se han ido reduciendo estos requisitos de consistencia e integridad, con el objetivo de obtener otros beneficios como son la escalabilidad y elasticidad.

Se ha ido popularizando el término *BASE* para describir este tipo de propiedades más optimistas:

Basic availability
: La base de datos funciona la mayor parte del tiempo.

Soft-state
: No tienen porqué garantizar la integridad de los datos, ni siquiera múltiples réplicas tienen que ser consistentes siempre.

Eventual consistency
: La base de datos muestra consistencia en un momento dado (tardía, en tiempo de lectura).

Estas bases de datos están entre las 10 tendencias tecnológicas más importantes del 2013 según Gartner. El Big Data estratégico es tendencia[24], y entre todas esas fuentes

de datos, cinco de las más ricas en Internet son Grafos : "*Social Graph, Intent Graph, Consumption Graph, Interest Graph and Mobile Graph*".

2.4 Bases de Datos Orientadas a Grafos

Una base de datos en grafo es un sistema de gestión de bases de datos con operaciones CRUD (*create, read, update, delete*) que expone un modelo de datos basado en grafos[25]. Habitualmente están optimizadas para tener un buen rendimiento transaccional y una alta disponibilidad.
Hay dos propiedades importantes que diferencian este tipo de tecnologías:

- El tipo de almacenamiento: Algunas de estas bases de datos usan un almacenamiento nativo del grafo, que está diseñado para almacenar y manejar grafos, pero no todas las bases de datos usan este tipo de almacenamiento nativo.
- El motor de procesamiento: Dependiendo de la tecnología escogida, presentará o no un motor de procesamiento sin índices y basado en adyacencia, este tipo de motor se conoce como motor de procesamiento nativo de grafos, y se distingue de otros que se basan en listas de adyacencias almacenadas en tablas.

Aquellas tecnologías que presenten un modelo nativo manejarán estructuras en grafo de una manera mucho más eficiente que otras que basen la adyacencia en JOINS a otras tablas.

Encontramos tres tipos fundamentales de modelos de datos:

- **Grafos de Propiedades**
  - Contienen nodos y relaciones entre ellos.
  - Los nodos contienen propiedades.
  - Las relaciones tienen nombre y están dirigidas.
  - Dichas relaciones pueden contener también propiedades, y siempre tienen un nodo origen y otro destino.
- **Hipergrafos**
  - Presentan híper-ejes que relacionan múltiples nodos.
  - Pueden tener múltiples nodos origen y destino.
  - Modelos visualmente más sencillos.
- **Tripletas**
  - Proceden del movimiento de la Web Semántica.

- Estructura de datos con Sujeto-Predicado-Objeto. En masa proveen sets de datos muy ricos en conocimiento.
- Altamente documentadas debido a que RDF (su lengua franca) y SPARQL son un estándar del W3C.
- No son bases de datos en grafo nativas, por lo que no soportan el modelo de adyacencia sin índices.
- Más lentas en búsquedas en grafo, se utilizan sobre todo en análisis de datos, debido a que es un entorno donde la velocidad no importa.

A pesar de que las bases de datos en grafo están diseñadas principalmente para rendir en recorrido de grafos y ejecutar algoritmos de teoría de grafos, es posible usarlas como soporte de persistencia detrás de un endpoint RDF/SPARQL. Algo que la API SAIL de Blueprints permite[26], proveyendo una interfaz RDF a diferentes bases de datos en Grafo. Esto supone un isomorfismo funcional entre las llamadas Triple-Stores y las bases de datos en Grafo, a pesar de que cada tecnología de almacenamiento subyacente está diseñada para una carga de trabajo diferente.

2.5 Neo4j: The World's Leading Graph Database

Dentro de las bases de datos orientadas a grafos de propiedades, destaca especialmente *Neo4j[27]*.

Esta base de datos se caracteriza por ser nativa, open source y una de las más populares dentro de su categoría. Sus funcionalidades más importantes son:

- Transacciones ACID: al contrario que muchas NoSQL, Neo4j soporta este tipo de transacciones.
- Alta disponibilidad: permite desplegar varias instancias en modo Maestro-Esclavo.
- Escala hasta miles de millones de nodos y relaciones.
- Búsqueda muy rápida en consultas de recorrido del grafo.
- Posee un lenguaje de consulta declarativo muy similar a SQL.

Neo4j brilla especialmente en aquellos modelos de datos que posean muchas interrelaciones entre entidades, ya que permite recorrer millones de pasos de un recorrido por segundo, mientras que en una base de datos relacional, cada paso implica un *join*. Para ilustrarlo mejor, citaremos un experimento realizado.

En el libro *Neo4j in Action[28]*, Jonas Partner y Aleska Vukotic realizan un experimento asombroso que demuestra la mejora de rendimiento que tiene una base de datos de grafos frente a una relacional, en escenarios donde se haga un uso intensivo de recorrido de grafos.

Se realizó una consulta tanto en MySQL como en Neo4j con una base de datos de 1000000 de usuarios. Llegados a una profundidad de 5 en el grafo, Neo4j terminaba en 2.132 segundos, mientras que en MySQL pasada 1 hora aún no había terminado.

El grafo etiquetado de propiedades de Neo4j, está conformado por distintas reglas que se describen a continuación:

- **Un Grafo contiene Nodos y Relaciones**

Los **Nodos** (Figura 2.12) en Neo4j representan generalmente entidades, pueden contener propiedades y relaciones, y además pueden ser etiquetados con cero o más etiquetas.

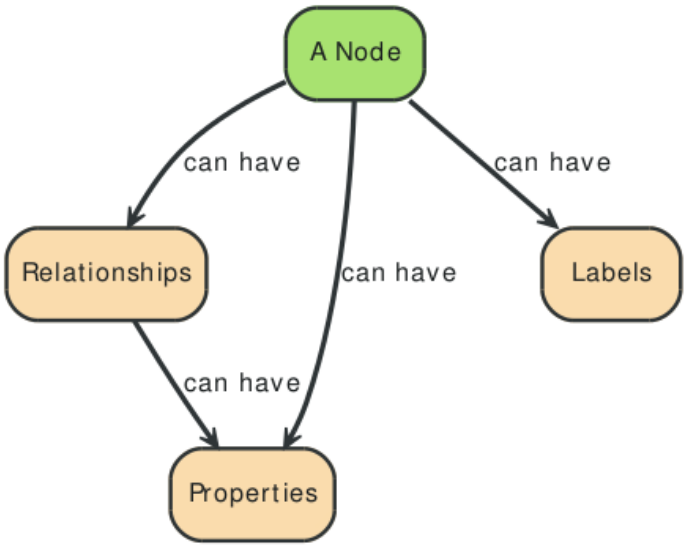

Figura 2.12 *Los nodos en Neo4j*

- **Las relaciones organizan el Grafo**

Le permiten organizar nodos en estructuras arbitrarias, de forma que el grafo puede asemejarse a una lista, un árbol, un diccionario, o una entidad compuesta que pueden

ser a su vez combinadas para crear estructuras más complejas. Además, las **relaciones** entre nodos pueden tener a su vez **propiedades** (Figura 2.13).

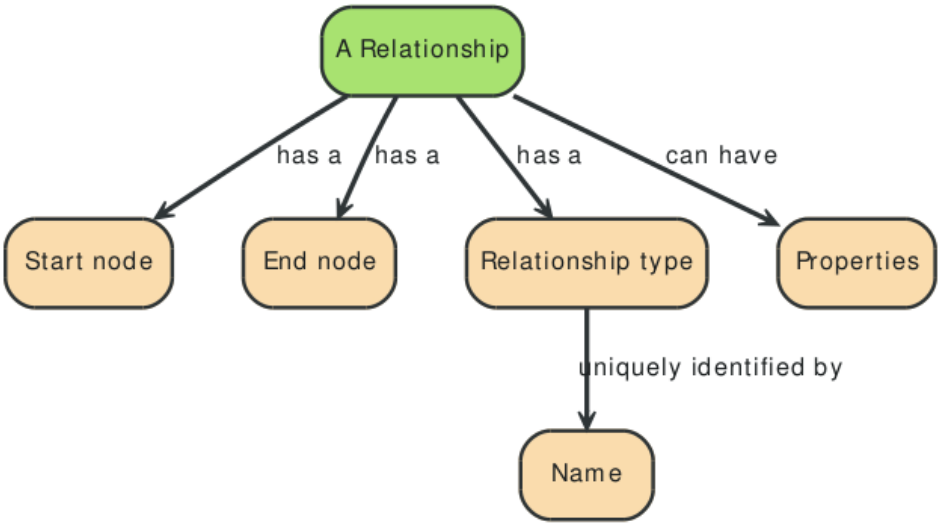

Figura 2.13 *Las relaciones en Neo4j*

Siempre tienen un Nodo origen y otro de destino, siempre están dirigidas, por lo que pueden ser relaciones entrantes o salientes (Figura 2.14).

Figura 2.14 *Relaciones entrantes y salientes en Neo4j*

Los nodos pueden tener relaciones referenciándose a sí mismas, formando bucles (Figura 2.14).

Figura 2.15 *Bucles en Neo4j*

Las Relaciones tienen un tipo asociado, que puede ser utilizado para representar de forma más rica los datos.

Tanto los nodos como las relaciones pueden tener propiedades (Figura 2.16). Estas propiedades son pares clave-valor, en los que la clave es un String. Los valores de las propiedades pueden ser, en cambio, cualquier tipo de datos primitivo, o un Array de dichos tipos.

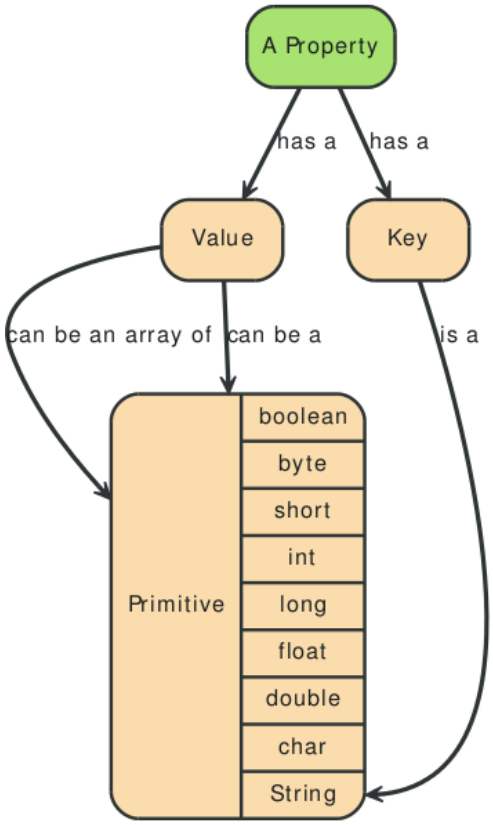

Figura 2.16 *Las propiedades en Neo4j*

- **Los nodos se agrupan mediante etiquetas**

Las **etiquetas** son una forma de agrupar los nodos dentro del grafo (Figura 2.17). Pueden ser utilizadas para aplicar restricciones a conjuntos del grafo para consultas, así como también para dotar de cierto esquema al modelo de datos. En la versión 2.0 Neo4j añade la funcionalidad de las etiquetas, esta función es experimental en el momento en que se redacta este Trabajo.

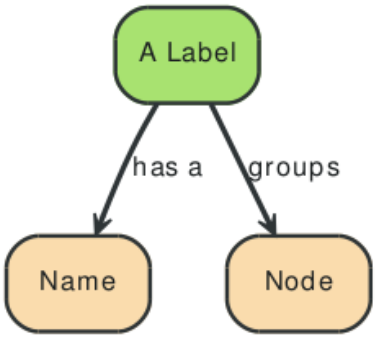

Figura 2.17 *Etiquetas en Neo4j*

Una etiqueta permite agrupar nodos en tipos; todos los nodos con una determinada etiqueta pertenecen al mismo grupo. Esto permite realizar búsquedas en grupos, en lugar de en todo el grafo, por lo que las búsquedas pueden realizarse de forma más eficiente. Sin embargo, las etiquetas pueden usarse para mucho más. Pueden añadirse y quitarse en tiempo de ejecución, por lo que pueden usarse para modelar estados temporales, como por ejemplo usuarios offline. Por otra parte, permite añadir un esquema al grafo, y hacer que sea compatible con modelización por Marcos, o Redes Semánticas.

- **El Grafo se consulta mediante recorridos**

Un **recorrido** es la forma en la que se consulta un grafo, navegando desde un nodo inicial a través de sus adyacentes, acorde a un determinado algoritmo (Figura 2.18).

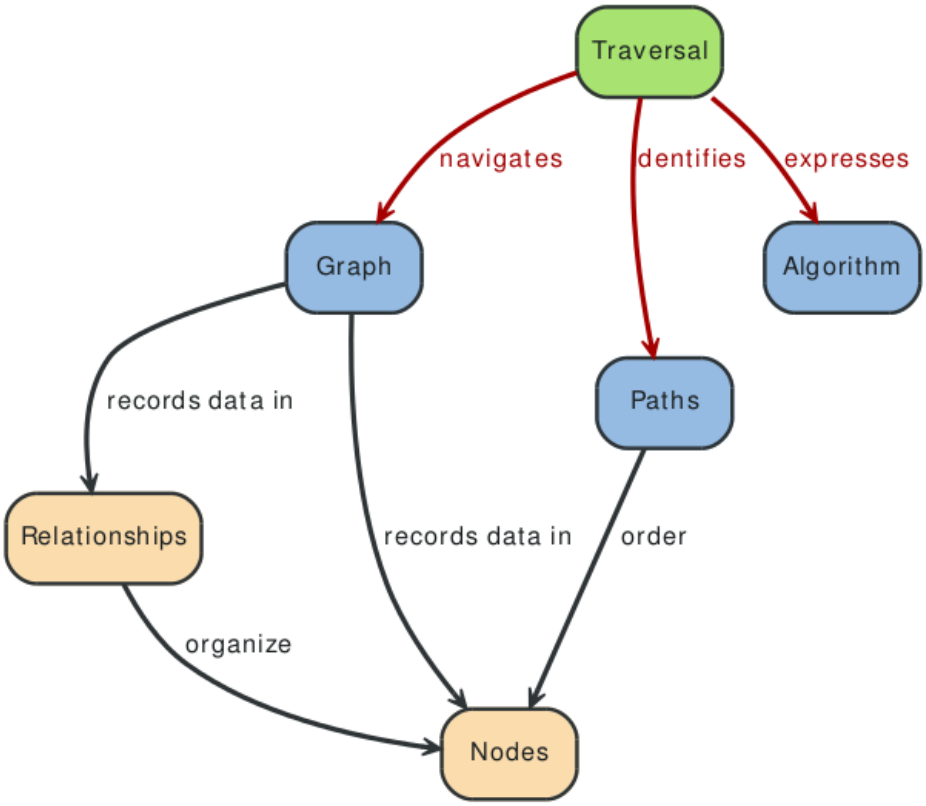

Figura 2.18 *Recorridos en grafos en Neo4j*

- **Los índices referencian Nodos o Relaciones**

Los **índices** permiten buscar información (Figura 2.19) en propiedades que son frecuentemente utilizadas, evitando tener que recorrer el grafo entero para acceder a dichos datos cada vez que se necesite.

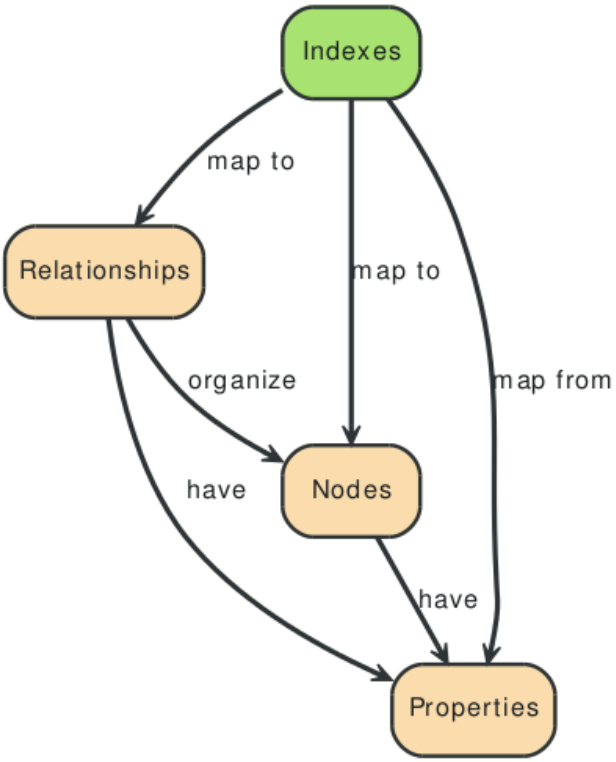

Figura 2.19 *Índices en Neo4j*

Las consultas y operaciones contra la base de datos se realizan principalmente vía la API Java de Neo4j, la API REST que ofrece el modo servidor, Gremlin, o Cypher, un lenguaje creado específicamente para esta base de datos.

Cypher es un lenguaje declarativo para realizar consultas en grafos. Permite realizar complejas búsquedas y modificaciones en *Neo4j*. Su sintaxis está pensada para que su lectura sea sencilla por humanos. Toma elementos de *SQL* y se estructura de la misma forma; otras características las toma en cambio de *SPARQL* – el lenguaje de búsqueda para *RDF* -. Por último, otras partes del lenguaje han sido tomadas de lenguajes como *Haskell* y *Python*. Como cualquier lenguaje declarativo y en contraste con los lenguajes imperativos, el problema se resuelve definiendo "Qué se quiere hacer", en lugar de "Qué hay que hacer".

2.6 Redes Sociales y la Web Semántica: Big-Linked-Open Data

> *"La ciencia se asemeja mucho a un árbol: en contra del pensamiento general, tanto los árboles como la ciencia crecen desde sus extremos, no desde su núcleo. En Informática, este pensamiento es particularmente cierto. El desarrollo más importante en Inteligencia Artificial en los últimos años ha tenido lugar dónde ésta se fusiona con la Web".*

Así inicia Peter Mika[29] su libro *Social Networks & Semantic Web*, en el que pone de manifiesto la importancia que puede tener la fusión de estas ramas tecnológicas[30]. Las redes sociales, suponen una fuente de información gigantesca, que los usuarios día a día hacen crecer y crecer. El concepto de Big Data se fusiona[32] con el de *Linked Data[31] (Figura 2.20)*, propio de la Web Semántica, si queremos que toda esta

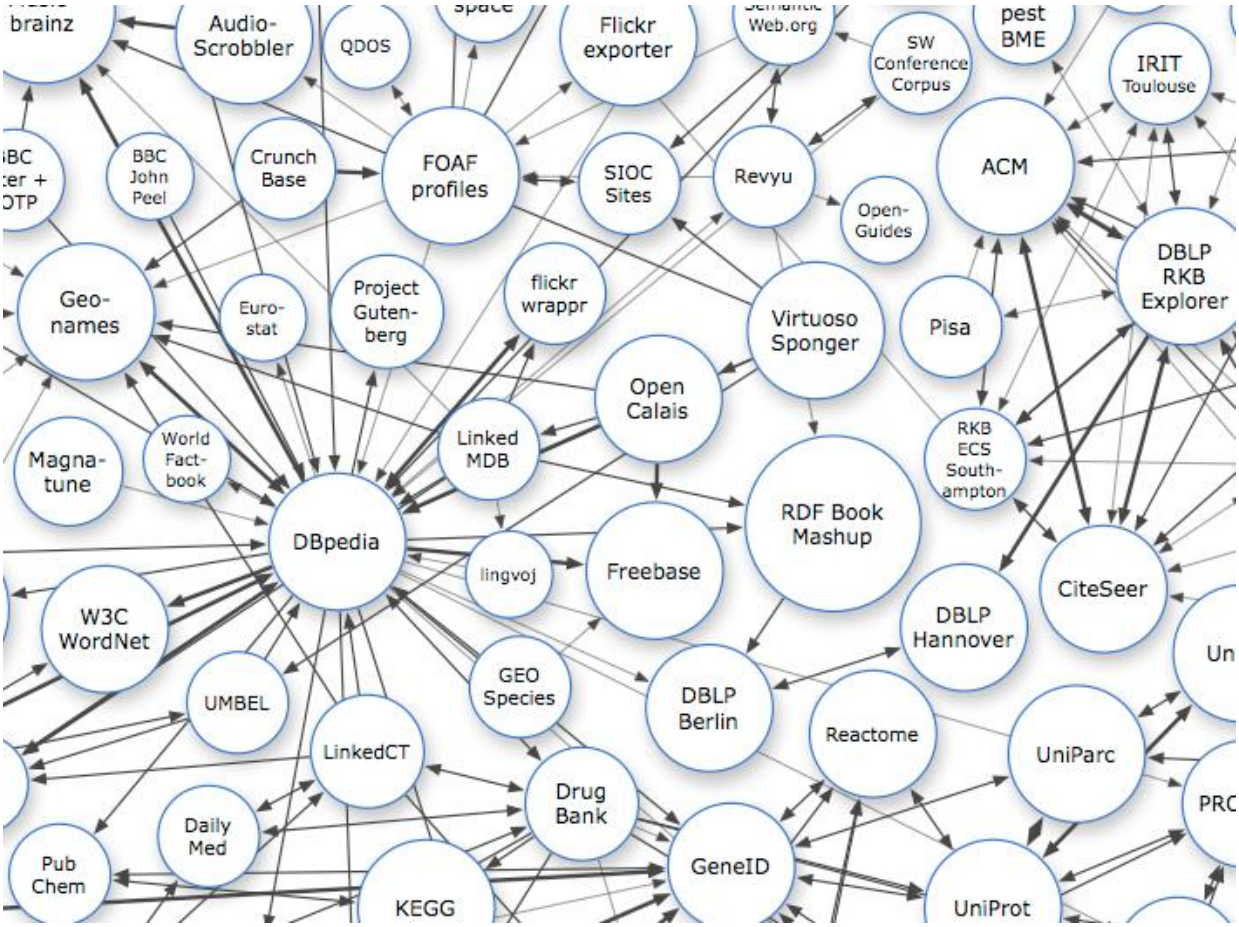

Figura 2.20 *Parte del diagrama del Proyecto Linking Open Data*

información se convierta en conocimiento útil. Asimismo, gran parte de estos datos pueden ser abiertos al público o a determinadas entidades, que deseen utilizar este conocimiento para sus propios intereses. Es lo que se denomina *Open Data*, un movimiento similar al del Software libre. Así, grandes empresas de Internet cuyos productos son gratuitos para los usuarios, basan su modelo de negocio en sus datos, que pueden ser consultados por otros a cambio de una contraprestación por su uso. Estos datos suelen estar representados en Grafos, así tenemos diferentes ejemplos, como el de Facebook Open Graph, Google Freebase[34], o Twitter Interest Graph, que permiten a terceros, utilizar su información abiertamente o bajo alguna contraprestación.

Es habitual organizar la información social en grafos, esto es así debido a que este modelo matemático representa de forma perfecta las interacciones sociales que se pueden producir en un sistema de este tipo. Tanto Google, Facebook, Twitter como LinkedIn, disponen de grafos en su núcleo, para representar de forma organizada sus datos.

Google implementa su Knowledge Graph[33] encima de Freebase, Wikipedia o la CIA World Factbook entre otras fuentes de datos, además de aquellos que va recolectando. Una base de conocimiento estructurada, por supuesto, en un Grafo.

Facebook implementa un middleware que se comunica con su infraestructura de datos montada sobre MySQL y Memcache[35].

Twitter desarrolló su propia base de datos NoSQL de Grafos, *FlockDB[36]* (con un grafo no nativo montado sobre *MySQL*), al igual que LinkedIn, que hizo lo mismo, con *Voldemort[37]* , su base de datos Clave-Valor que transforma relaciones entre grafos.

Estos sistemas que tienen un fuerte componente social, presentan fuentes de datos evolucionadas, que son *de facto* bases de conocimiento[38] implementadas con tecnologías dispares (Figura 2.21).

Figura 2.21 *Diferencias entre datos, información y conocimiento*

Un grafo, permite estructurar los datos de una forma inteligible, no sólo como texto: "*Things not Strings*"[39]. La información catalogada en estos sistemas de información, por tanto, es entendible y consumible no solo por humanos, sino también por máquinas.

Algunas de las principales bases de conocimiento disponibles en Internet son:

- Freebase
- DBpedia
- True Knowledge
- Wolfram Alpha

Estos sistemas tratan de representar el conocimiento humano en una base de conocimiento, pero los campos de aplicación son casi infinitos.

Sobre estos sistemas de información, se pueden aplicar técnicas de Minería de datos por lo que estamos hablando de sistemas muy ricos para la representación de datos.

2.7 Mobile Cloud Computing

Las razones por las que la computación móvil es vista como el futuro de la computación en general, son sencillamente que los humanos, somos seres inherentemente móviles. Yishan Wong, CEO de Reddit y actual Director de Ingeniería de Facebook, comenta estas razones[40] y añade que casi cualquier herramienta que en el pasado sirvió de utilidad al ser humano, ha sido móvil.

Un ordenador es una herramienta increíblemente útil, pero si podemos hacerla transportable, estaremos haciéndola mejor y tendrá un mayor alcance.
Durante los últimos años, la industria ha estado vendiendo computadores móviles, al tiempo que los usuarios creían que estaban comprando teléfonos. Este hecho, ha incrementado las posibilidades de todas aquellas aplicaciones que funcionen a través de Internet. Además los Smartphones, reemplazan a otros dispositivos tradicionales como las cámaras digitales, los dispositivos GPS, radios o dispositivos MP3, y una larga lista.

Uno de los indicadores más claros de que la computación móvil está para quedarse, es que la tendencia actual es la denominada "*Mobile First",* que consiste en desarrollar primero pensando en la aplicación móvil, y después la de escritorio. De haberse concebido primero en escritorio, es posible que algunas funcionalidades para la aplicación no hubiesen surgido.

El panorama actual está dominado principalmente por Samsung y Apple en el aspecto Hardware, y Android e iOS, de Google y Apple respectivamente, en el aspecto Software.

El futuro en cambio, parece pasar por el desarrollo de aplicaciones en HTML5: el año pasado Sundar Pichai, Director de los proyectos de *Google Chrome, ChromeOS y Google Apps*, pasó a ocuparse también del desarrollo de Android. Existen opiniones que relacionan a Android con ChromeOS en una más que posible fusión de ambos, y un cambio de dirección hacia ChromeOS, y con él las aplicaciones móviles HTML5, dejando de lado las nativas, que ganaron importancia gracias al éxito de iOS que las popularizó, y que ahora que empieza a dejar de ser una amenaza para Android empezarían a perder su momento. Este es el modelo que Mozilla también pretende seguir, con su nuevo Sistema Operativo FirefoxOS, basado íntegramente en HTML5 y Javascript.

En la parte Software, Android e iOS, son los reyes actuales en el mercado móvil. Se estudiará un poco más en detalle estos dos Sistemas Operativos, para poder ver así, sus principales ventajas y desventajas.

2.7.1 iOS

iOS[41] es el sistema operativo de Apple para sus Smartphones y Tablets: iPhone e iPad, y actualmente se encuentra en su versión 7 (Figura 2.22). Deriva del Mac OS X, un sistema operativo basado en UNIX. Sus aplicaciones se programan principalmente en el lenguaje *Objective-C* usando el SDK de Apple para iOS, el IDE Xcode, y el Interface Builder para el Frontend de las aplicaciones. Sus aplicaciones se distribuyen mediante la Apple Store.

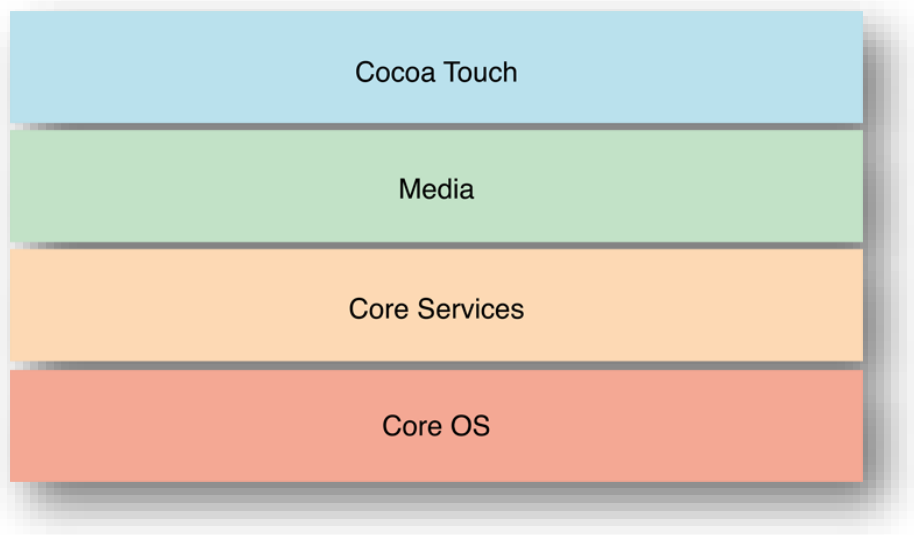

Figura 2.22 *Arquitectura Software de iOS (Fuente: iOS Developer Library)*

2.7.2 Android

El Sistema Operativo Android[42] es el otro actor principal en la escena móvil actual. Fue desarrollado en un principio por la empresa Android.Inc, apoyado económicamente por Google durante años, y finalmente adquirido en 2005 por la misma.

Android está construido sobre un Kernel modificado de Linux (Figura 2.23), su desarrollo actualmente se realiza por el Android Open Source Project, liderado por Google. Actualmente se encuentra en su versión 4.4 KitKat. El lenguaje de programación es principalmente *Java* usando el SDK de Android, que incluye un emulador de un entorno de ejecución para depuración y pruebas. Las aplicaciones se

distribuyen a través de la tienda *Google Play* y otros canales de distribución, como Aptoide.

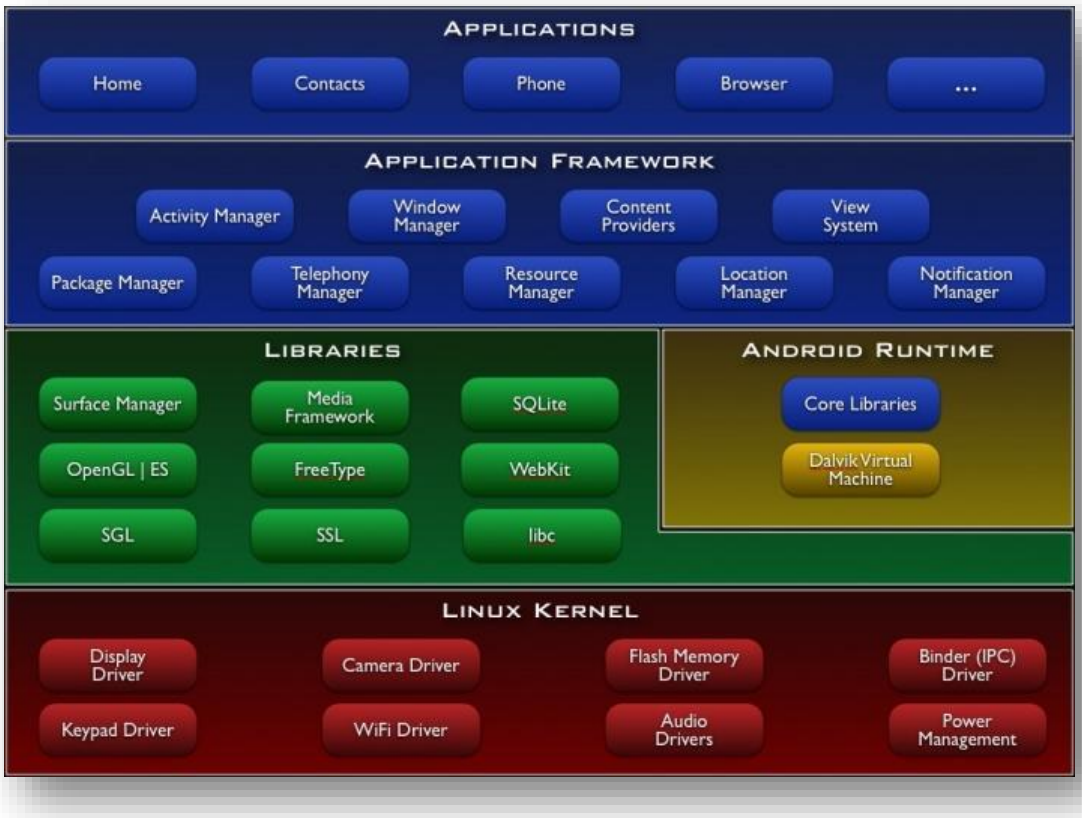

Figura 2.23 *Arquitectura Software de Android (Fuente: Android Developers)*

2.8 Tendencias Web del 2014

Desde hace relativamente poco tiempo se vienen considerando ciertas tecnologías para el desarrollo web como algunas de las mejores opciones en este campo. Javascript, Node.js, AngularJS o Bootstrap están a la orden del día en multinacionales punteras en tecnología y *Startups* de *Silicon Valley*. En este apartado se describen aquellas que se han usado para el desarrollo de CloudRoom, ahondando en su historia, y detalles técnicos para comprender mejor el porqué de su elección.

2.8.1 Javascript

The World's Most Misunderstood Programming Language [43]. Así definió Douglas Crockford a **Javascript**, un lenguaje interpretado que surgió como un plug-in del navegador Netscape. Durante mucho tiempo fue vilipendiado y poco respetado[44]. La alta ofuscación del código, que llega a ser poco legible; así como también partes del lenguaje que contienen inconsistencias, y permiten (si no se conoce bien el lenguaje) cometer errores de tipado son algunas de las razones por las que ha sido criticado. Además, la implementación de la herencia es bastante compleja comparada con otros lenguajes. En esta dirección han surgido lenguajes que pretenden abstraer al programador del uso de las "malas partes" de Javascript. Lenguajes como CoffeeScript o TypeScript, que presentan una sintaxis más sencilla, que evita dichas partes negativas y se compilan directamente a Javascript. No obstante, con la llegada de las aplicaciones AJAX como Gmail o Google Maps, esta concepción empezó a cambiar. El hecho de ser un lenguaje interpretado también le ha relegado a una segunda fila como lenguaje de desarrollo de aplicaciones complejas. Sin embargo, y a partir del año 2000, los avances por parte de Google y Mozilla en optimización de sus motores de Javascript – *V8* y *Tracemonkey* respectivamente – han logrado aumentar el rendimiento de las librerías de este lenguaje, llegando a acercarse, y en algunos casos a superar[45] el rendimiento de aplicaciones construidas con lenguajes compilados.

Más tarde, en 2009 un ingeniero llamado Ryan Dahl aprovechó el código fuente de Chrome y su V8 para utilizarlo en un sistema fuera del navegador, y anunció una nueva plataforma de desarrollo del lado del servidor para Javascript llamada **Node.js**. Sus facilidades para la entrada/salida no bloqueante han conseguido hacerlo muy popular entre muchas nuevas startups.

Hoy en día, se dice que Javascript es la lengua franca de Internet, y probablemente el lenguaje de programación más utilizado del mundo.

2.8.2 Node.js

Muchos lenguajes tradicionales manejan un modelo de concurrencia por hilos de ejecución (*threads*), que hacen uso de recursos compartidos, provocan condiciones de carrera en determinadas situaciones, y con los que es realmente complejo de diseñar e

implementar la concurrencia. Además la eficiencia de los hilos de ejecución está lejos de la que pueden ofrecer otros mecanismos.

Node.js es una plataforma construida sobre la máquina virtual **V8** para JavaScript de Google Chrome. Permite de forma ligera construir aplicaciones en red escalables[46] a través de un modelo de Entrada/Salida no bloqueante orientado a eventos. Node se comunica con el Sistema Operativo a través de llamadas al Kernel, define eventos, los cuales harán una llamada de vuelta (*callback*) en el momento que se produzca tal evento. Esos *callbacks* utilizan una pequeña zona de memoria aislada y estanca, y ejecutan ahí sus operaciones. Esto contrasta con el modelo de concurrencia ya comentado, que utiliza hilos del sistema operativo, y evitan condiciones de carrera usando mecanismos de sincronización, como los cerrojos. En Node, la concurrencia es asíncrona, nada se bloquea, nunca. No se pueden producir interbloqueos, ni inanición, puesto que nada está bloqueado. Por consiguiente, muestra una mayor eficiencia en comparación con el otro modelo en situaciones como la de una alta carga de un Servidor, y en general en sistemas que hagan un fuerte uso de la Entrada/Salida. Razones estas, que lo hacen ideal para su uso en sistemas distribuidos.

2.8.3 Middlewares y Frameworks

Existen decenas de Frameworks y plataformas para Javascript, y otros tantos middlewares para estos. Tanto para el lado del cliente como el lado servidor. Estos a su vez, tienen dependencias de otros middlewares que ya incluyen implícitamente.
En esta subsección se habla de aquellos utilizados en CloudRoom.

2.8.3.1 Express.js

Express.js[**47**] es un framework para Node.js que provee al programador de una serie de funciones de más alto nivel para la implementación de aplicaciones web. Es el framework de referencia para realizar servidores y APIs REST. Su principal virtud es la simplicidad de sus operaciones. Algunas aplicaciones comerciales que lo utilizan son el nuevo MySpace o Klout.

2.8.3.2 Restler

Restler[48] es un middleware para realizar clientes REST en Node.js, abstrae la mayoría de la complejidad de crear y usar directamente la librería HTTP de Node.

2.8.3.3 Bcrypt-nodejs

Basada en el algoritmo criptográfico Blowfish diseñado por Bruce Schneier. Esta librería[49] nativa de Javascript para node, permite cifrar cualquier contraseña con una longitud de hasta 448 bits.

2.8.3.4 Cors

Dentro de Express.js podemos añadir middlewares para facilitarnos ciertas tareas comunes, como la de habilitar el *Cross-Origin Resource Sharing*, gracias a este middleware.

2.8.3.5 Passport.js

Para el proceso de autenticación, y la gestión de las sesiones de los usuarios, **Passport.js**[50] nos abstrae de lidiar con los prolegómenos que estas tareas tienen. Además, nos permite utilizar diversas estrategias de gestión de sesión y autenticación o crear las nuestras propias. Los mecanismos más destacables son: tradicional (con nombre de usuario y contraseña), OpenID y oAuth (pudiendo seleccionar entre proveedores como Google, Facebook o Twitter).

2.8.3.6 Connect-Redis

Este middleware[51] abstrae al programador de implementar los mecanismos de conexión con la base de datos **Redis**. Permite de forma transparente conectar una o múltiples instancias de Redis.

2.8.3.7 AngularJS

Hoy en día, se construyen aplicaciones web impresionantes, sin embargo, la complejidad de su construcción no lo es menos. En Google, crearon **AngularJS**[52] después de haber sufrido con la creación de aplicaciones web colosales como Gmail o Google Maps.

Angular, no es sino un framework Modelo Vista Controlador (MVC) que proporciona una capa de abstracción al programador sobre el *Document Object Model* (DOM), y además permite organizar el código del Front-End de una forma estructurada, facilitando la mantenibilidad, y logrando que las decisiones de diseño de la lógica de la aplicación sean más naturales.

2.8.4 Responsive Web Design

Quizás la mayor de las tendencias web en el último año y del que acaba de comenzar, es el diseño de interfaces web adaptativas; un término que en inglés se denomina *Responsive Web Design[53]* – en adelante RWD -. Este concepto se hace imprescindible hoy en día para desarrollar prácticamente cualquier aplicación multidispositivo orientada a la web; puesto que el primer paso en el desarrollo Front-end de un sistema es su interfaz web. Una interfaz desarrollada mediante RWD, se adapta a la pantalla de aquel dispositivo en el que se esté visualizando mediante una serie de reglas de resolución de pantalla, definidas gracias a nuevas funcionalidades añadidas por CSS3. Esto es de gran valor, puesto que no solo proporciona interfaces más modernas y atractivas para el usuario, sino que añade un plus de productividad en el desarrollo de cualquier aplicación multidispositivo, que de otra manera, necesitaría lanzarse al desarrollo de aplicaciones nativas para móviles. Sin contar con que en ordenadores de escritorio, portátiles y similares, nunca estaría optimizado para la mayor parte de dispositivos.
Las tecnologías que hacen posible esto en CloudRoom se describen en los siguientes subapartados.

2.8.4.1 HTML5

Hiper Text Markup Language (HTML) es el lenguaje estándar de representación de la web. Una página web, es un documento HTML.

Formado por etiquetas y texto plano, el documento se organiza en una estructura lógica de árbol. El Domain Object Model (DOM), es el encargado de definir esa estructura lógica, y ofrecer una interfaz de programación a través de la cual poder hacer modificaciones de forma dinámica del documento.
Su última versión **HTML5**[54], supone cambios importantes con respecto a la anterior, como son:

- Multimedia: Por primera vez establece un estándar para introducir video y audio.
- Gráficos: Mediante el uso de una nueva etiqueta se permite la referencia de gráficos vectoriales, además del uso de gráficos 3D/2D
- Aplicaciones web: Facilita el desarrollo de aplicaciones web gracias a nuevas funcionalidades como:
    - Almacenamiento local de datos
    - Acceso a ficheros locales
    - Acceso a una base de datos SQL local
    - Cache de aplicación
    - Workers Javascript
    - XHTMLHttpRequest 2
- Semántica: Nuevas etiquetas permiten añadir semántica, tales como Cabeceras, pies de página, menús, secciones y artículos.
- Formularios: Nuevos elementos, atributos, tipos de entrada de datos, con validación automática de campos.
- Integración con CSS3

2.8.4.2 CSS3

Cascading Style Sheets son las siglas de las hojas de estilo externas en las que se almacenan los estilos introducidos en HTML4. **CSS3**[**55**] es su última versión, y está aún pendiente de ser aprobado como recomendación oficial del W3C, sin embargo, al igual que HTML5, muchos navegadores ya implementan funcionalidades de esta versión.
Mediante el uso de hojas de estilo externas podemos ahorrar mucho trabajo, puesto que editando un único archivo nos permite cambiar el estilo de un documento HTML. De otra manera, habría que aplicar a mano los estilos, en cada zona que deseáramos, ofuscando el código HTML y dejando un documento poco mantenible.

2.8.4.3 Bootstrap

Twitter Bootstrap[56] es una colección de herramientas de software libre para la creación de sitios y aplicaciones web. Es usado por la NASA, y Coursera, entre otras organizaciones. Es un framework muy poderoso para diseño de Front-end *Mobile first*, debido a que proporciona de manera casi automática un diseño RWD de la interfaz gráfica, permitiendo desarrollar primero para dispositivos móviles, y que ese diseño sea reutilizable para la interfaz de escritorio de mayor tamaño. Esto funciona gracias a un sistema de malla (*Grid System*) que organiza los elementos que se muestran por pantalla en 12 columnas, dichas columnas se auto organizan en función del tamaño de la pantalla en tiempo real. Funciona por tanto con cualquier tamaño de pantalla.
Fue desarrollado en Twitter en un intento de utilizar un estándar para el desarrollo del Front-end, y evitar así problemas de conflictos entre desarrolladores e inconsistencias entre librerías.
En su núcleo, Bootstrap es puramente CSS, pero está construido con *Less*, un lenguaje de declaración de hojas de estilo. Ofrece más flexibilidad de desarrollo, aportando funcionalidades como declaraciones anidadas, variables, mixins, operaciones, y funciones de coloreado. Posteriormente, *Less* se compila en CSS.
Aunque Bootstrap es en el fondo CSS puro, cuando se compila desde Less se obtienen dos beneficios fundamentales:

1- Implementar con Bootstrap es tan sencillo como incluirlo en el código HMTL y empezar a utilizarlo.
2- Una vez compilado, solamente contiene CSS. Se asegura de esta forma que no hay imágenes superfluas, Flash o Javascript.

Bootstrap está diseñado para proveer al desarrollador de recursos típicos para solucionar problemas a los que se enfrenta un desarrollo de Front-End en cualquier proyecto. Además es lo suficientemente flexible como para conseguir diseños únicos en cualquier sitio que se pretenda diseñar.

Capítulo 3
Planteamiento del Problema

El problema al que nos enfrentamos es la concepción, diseño y construcción de un sistema *Software as a Service* de gestión de cursos abiertos y masivos online (MOOC) con funcionalidades sociales y semánticas, accesible desde dispositivos móviles.

La idea de una plataforma de MOOCs, es muy novedosa en su concepto. Existen pocos sistemas de este tipo en la actualidad, algunos en pleno desarrollo, en constante evolución, y no hay mucha experiencia ni conocimiento sobre ellos. No mucha más información tenemos de las redes sociales. Con infraestructuras tremendamente complejas, ignoradas por muchos profesionales informáticos, en algunos casos disponen de auténticos silos de información, que debe ser procesada, almacenada y analizada.

Construir un producto de este tipo desde sus cimientos no es tarea fácil, conlleva una gran inversión de tiempo en investigación, análisis de las funcionalidades que debería presentar el producto, diseño de la arquitectura de este tipo de plataformas, estudio de las tecnologías adecuadas, o despliegue de este tipo de sistemas en alta disponibilidad. Requiere además una curva de aprendizaje relativamente alta de todas estas tecnologías. Y añade una complejidad adicional, ya que al ser un sistema con una alta incertidumbre en sus requisitos, se desconoce realmente si lo que se pretende diseñar en una fase temprana, va a funcionar finalmente de esa manera.

La segunda cuestión que convenía resolver era como abordar el diseño de una red social. Un tipo de sistemas software relativamente novedosos y del que no hay mucha documentación. Tras un tiempo de investigación, se descubrió una tecnología que encajaba perfectamente en nuestra idea: una base de datos de grafos.

El uso que las redes sociales hacen de los datos es en algunos casos descomunal. Un volumen descomunal de datos, con información muy variada, estructurada y desestructurada, que se crea a una gran velocidad, y que genera como se ha visto en la investigación previa, un valor muy importante para terceros, pero que requiere una veracidad.

Esas son las llamadas 5 "Vs" del *Big Data*, las que determinan que estamos ante un sistema con un modelo de datos *Big Data*.

Una red social es un grafo, una taxonomía perfecta para ser almacenada en una base de datos de grafos. Y Neo4j brinda una forma natural de almacenar información semiestructurada, en forma de grafo, con la semántica que ello añade a nuestros datos, y las ventajas que una base de datos NoSQL ofrece a un sistema distribuido.

3.1 Análisis de Requisitos Software

En esta sección se detallan todas aquellas técnicas aplicadas y sus resultados para definir el producto a construir y extraer requisitos software.

Se han utilizado técnicas ágiles de Ingeniería del Software como las de Historias de usuario, así como también técnicas de usabilidad de Interacción Persona Ordenador.

3.1.1 Mapa de Navegación

Para estudiar las funcionalidades y extraer las ideas del concepto del producto, se han utilizado técnicas de *Brainstorming*, y *Mindmapping*. En esta sección, se detallan los resultados de dichas técnicas, que iniciaron el proceso de análisis formal de requisitos del software.

El mapa de navegación, describe las posibles ventanas que la aplicación mostraría, detallando el flujo de interacción entre cada una de ellas, así como cada una de las posibles bifurcaciones entre ventanas (Figura 3.1).

Todo el mundo tendría acceso a una página principal pública, desde la que se mostrarían cursos ofertados en la plataforma, como manera de atraer a posibles

usuarios interesados en esos cursos sin registro previo. Por otro lado, el existiría una página privada principal que daría acceso directo a toda la funcionalidad de la plataforma.

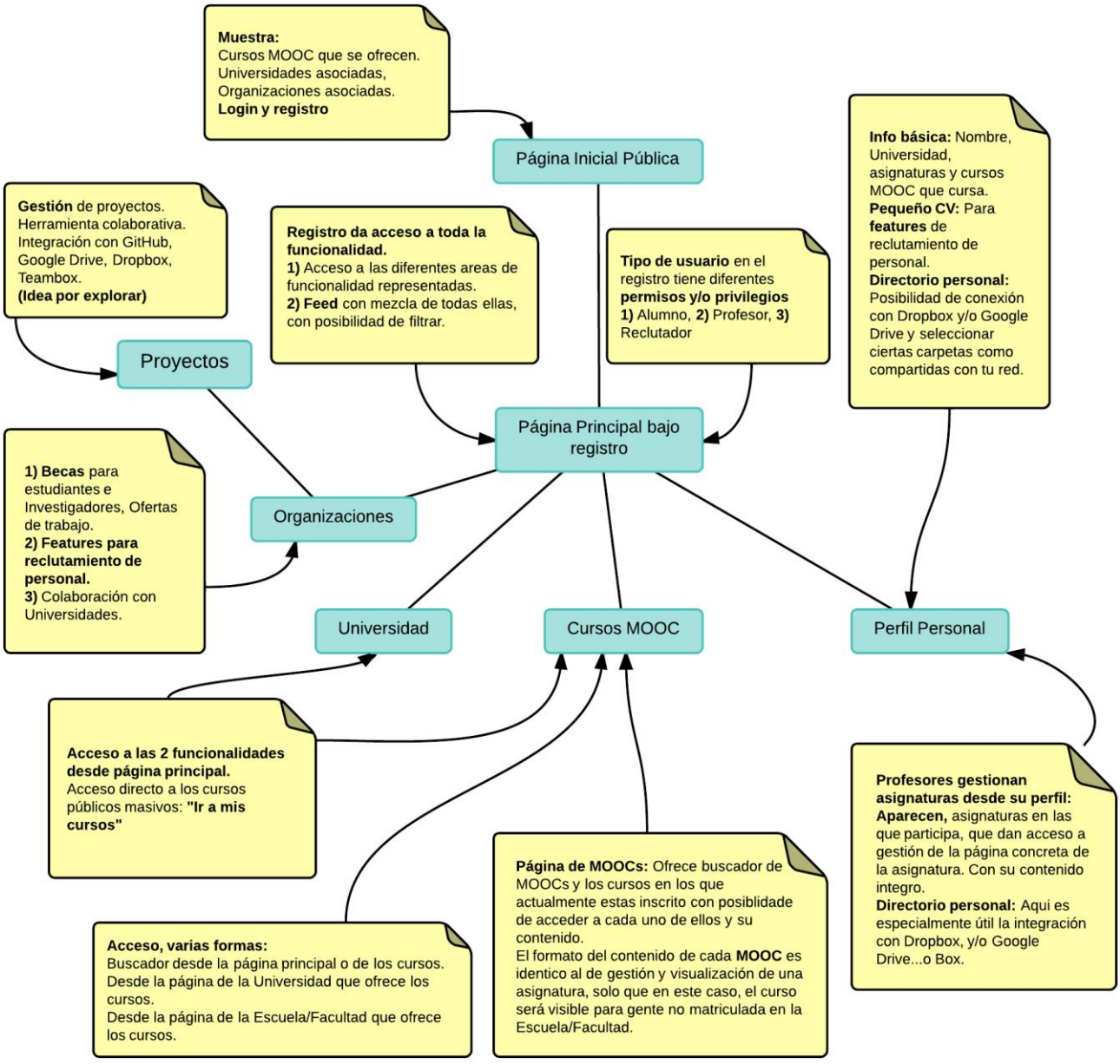

Figura 3.1 *Mapa de navegación de CloudRoom*

3.1.2 Especificación del Contexto de Uso del Sistema

En esta subsección se especifica el contexto de uso del sistema realizado en los primeros pasos del Proyecto.

La plataforma puede utilizarse principalmente en tres dispositivos diferentes:

- PC
- Tablet
- Smartphone

Al ser una aplicación centrada en la gestión de cursos, y de e-learning, se debe facilitar el acceso a las tareas, y favorecer una lectura cómoda. Además, se identifican dos contextos principales de uso: Alumno y Profesor.

3.1.2.1 Tareas de Alumno

1. Registro de usuario
2. Login de usuario
3. Mensajería entre todos los usuarios
   - Chat con aquellos directamente relacionados (Compañeros de curso, amigos o profesores)
4. Acceso al perfil de usuario
   - Personalización del perfil de usuario
   - Compartición de contenido público o privado vía publicaciones
   - Compartición de contenido vía Dropbox
5. Buscador, y Recomendador de cursos
6. Gestor de cursos matriculados
7. Acceso a perfiles de Universidades dentro de la plataforma
   - Feed con últimas noticias
   - Lista de cursos que ofrece
8. Acceso a la clase virtual
   - Acceso al contenido del curso
   - Acceso a clases grabadas en video
   - Acceso a foros del curso
   - Realización de tests
   - Realización de subidas de entregas

3.1.2.2 Tareas de Profesor

Mismos permisos que Alumno, pero además:

1. Creación de Curso
   - Personalización de la estructura de página del curso
   - Personalización del contenido del curso
   - Creación de Tests
   - Creación de tareas para entregas
   - Creación de correctores para entregas
2. Modificaciones de cursos
   - Gestión del contenido del curso
   - Gestión de tests y tareas

3.1.3 Esquema de Uso del Sistema

Aquí se trazan los pasos que un hipotético usuario podría seguir al acceder al sistema (Figura 3.2).

1. **Se permite registro de usuarios bajo cualquier dirección de email**
2. **En el registro de usuario se pide:**
   - Nombre completo real
   - Dir. Email (comprueba enviando mensaje para finalizar registro)
   - Contraseña
3. **Usuario dispone de la siguiente información para personalizar:**
   - Nombre y apellidos
   - Privacidad de su perfil
   - Los siguientes campos son opcionales:
     - Localización
     - Foto de perfil
     - Fecha de nacimiento
     - Descripción personal
     - Direcciones personales en Internet:
     - Página web personal
     - Cuentas en redes sociales:
     - Twitter
     - Facebook

- LinkedIn

4. **Una vez autenticado dentro de la aplicación:**
   - Página principal que contiene:
     - Cursos en los que está matriculado el alumno
     - Universidades y escuelas en las que está involucrado el alumno.
     - Página del perfil personal del usuario
     - Si es alumno
     - Si es profesor
     - Acceso a la página principal de dicha Universidad en CloudRoom que contiene:
     - Feed con noticias importantes.
     - Cursos que ofrecen las distintas Escuelas/Facultades de dicha Universidad.
5. **Página principal del curso contiene:**
   - Menú personalizable por el creador del curso que permite filtrar el contenido del panel de información del curso:
     - Inicio: Contiene últimos anuncios y noticias producidos en el curso.
     - Tareas: Lista que se va actualizando con todo el material que se va demandando al alumno
     - Calendario detallado del curso
       - Clases en video: Lista actualizable con vídeos de cada lección.
     - Lecciones del curso: Muestra el contenido íntegro de esa lección, con apuntes, videos, y deberes asociados a dicha lección, además de cualquier información o indicaciones que se quiera dar sobre esa lección al alumno.

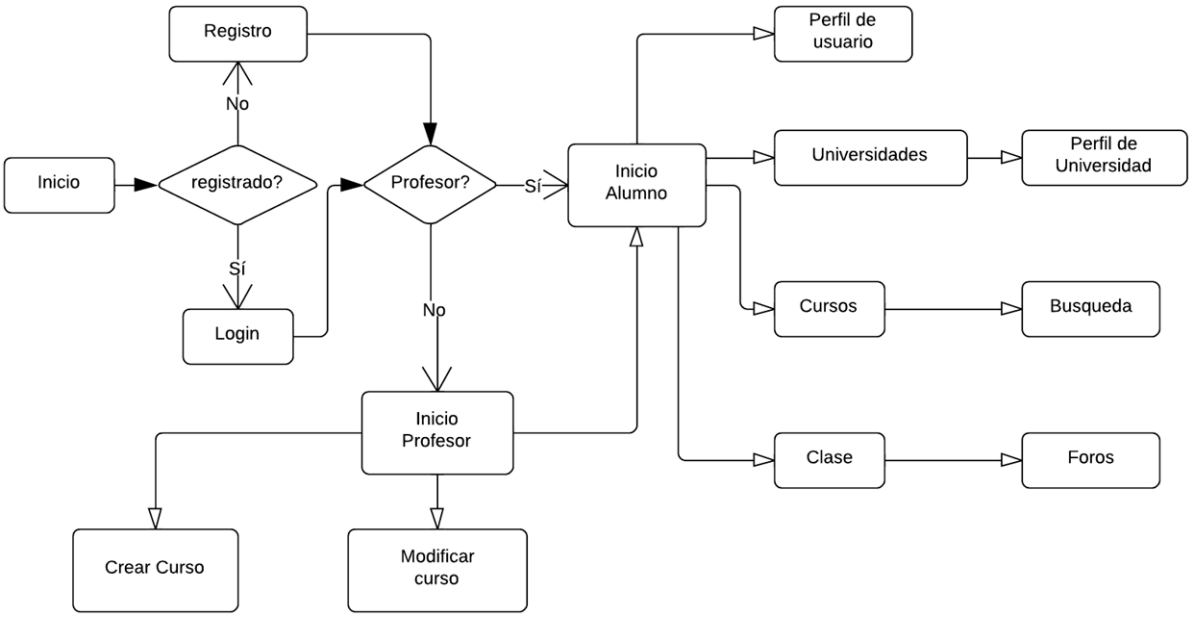

Figura 3.2 *Diagrama de flujo de uso*

3.1.4 Historias de Usuario

Para el análisis de requisitos, se han utilizado también historias de usuario ágiles en lugar de casos de uso ya que al no estar claros los requisitos del sistema a construir se tenía que lidiar con una alta incertidumbre, y estas proporcionan una manera menos costosa de analizar requisitos.

3.1.4.1 Funcionalidad básica

1) Registro:

1.1) Como alumno quiero poder registrarme y acceder con mis datos personales en la plataforma.

1.2) Como empresa quiero poder registrarme en la red social como una organización más e interaccionar con las Universidades, Comunidades y demás miembros de la red social.

1.3) Como profesor deseo poder registrarme para ofrecer recursos, administrar asignaturas, y publicar contenido de interés para la comunidad universitaria o para la red social.

2) Página personal:

2.1) Como alumno quiero poder personalizar mi página personal (visible al público).

2.2) Como usuario deseo poder compartir recursos importantes (apuntes, exámenes, etc.) a través de Google Drive.

2.3) y/o Dropbox.

3) Página principal de la aplicación:

3.1) Como alumno quiero poder consultar la información de las asignaturas de las que estoy matriculado de forma preferente.

3.2) Como alumno quiero una página de inicio en la que destaquen las últimas noticias filtradas por fecha y/o importancia sobre cualquier recurso al que esté suscrito.

3.3) Como usuario quiero disponer de un buscador para encontrar usuarios, universidades u organizaciones en la red social.

4) Mensajería y Contacto entre usuarios:

4.1) Como alumno deseo poder contactar de forma sencilla desde la plataforma con cualquier otro usuario ya sea alumno o profesor.

5) Notificaciones

5.1) Como alumno quiero tener en todo momento visible un icono de notificaciones en tiempo real de mensajes, eventos, e interacciones importantes en mi red.

6) Organizaciones

6.1) Como alumno quiero poder visitar la página oficial de las organizaciones que se inscriban en la red social.

6.2) Como alumno quiero poder contactar con usuarios pertenecientes a dichas organizaciones.

7) Empleo

7.1) Como alumno quiero poder consultar desde un único Feed las últimas ofertas de empleo publicadas.

7.2) Como empresa quiero poder ofrecer de forma sencilla ofertas de empleo dirigidas a estudiantes y profesores de todas las entidades pertenecientes a la red social.

3.1.4.2 Funcionalidad adicional

1) Recomendaciones

1.1) Como alumno quiero tener sugerencias personalizadas de cursos, empleos u otros.

2) Acceso multidispositivo

2.1) Como usuario quiero poder acceder a las funciones principales de la aplicación desde mi Smartphone (Android, iPhone, Windows Phone...).

2.2) Como usuario quiero poder acceder desde mi Tablet (iPad, Android...).

3.2 Especificación de Requisitos Software

A continuación se explican de forma detallada cuales son las funcionalidades que el sistema debe cumplir. Los requisitos han sido extraídos en la fase de análisis, sin embargo, no todos eran objetivo en este trabajo, pues implementarlos en el tiempo dado sería materialmente imposible. No obstante, se detalla la lista de todos los requisitos analizados en una ERS formal, para valorar todo el trabajo de análisis que se ha realizado.

1. Introducción

Esta especificación de requisitos software ha sido elaborada con el objetivo de detallar el funcionamiento del sistema CloudRoom. Para ello se han seguido las directrices dadas por el estándar "*IEEE Recomended Practice for Software Requirement Especifications ANSI/IEEE 830 1998*".

1.1. Propósito

El objeto de esta especificación es definir de manera clara y precisa todas las funcionalidades y restricciones del sistema a construir.

1.2. Ámbito del Sistema

El sistema a desarrollar tiene como objeto fundamental la gestión y acceso a cursos online de cualquier tipo, de manera abierta y masiva. Permitiendo la creación de cursos y su consumo por parte de cualquier persona. El sistema recibe el nombre de *CloudRoom*.

1.3. Definiciones, Acrónimos y Abreviaturas

AWS *Amazon Web Services*
MOOC Massive Open Online Courses

1.4. Bibliografía

IEEE Recommended Practice for Software Requirements Specification. ANSI/IEEE std. 830, 1998.

2. Descripción General

CloudRoom pretende ser un plataforma cloud social, semántica que ofrece cursos masivos abiertos al estilo MOOC (Massive Online Open Course) para cualquier persona del mundo, con contenidos personalizados, en cualquier dispositivo (smartphone, tablet, desktop ...), y certificados de completitud y asistencia para sus alumnos.

2.1. Perspectiva del Producto

El sistema interactuará con diversos sistemas informáticos externos, todos ellos aplicaciones SaaS. Son: Navegador web, Sistemas de almacenamiento en nube como Dropbox o Google Drive, y redes sociales como Facebook, Twitter, LinkedIn (Figura 3.3).

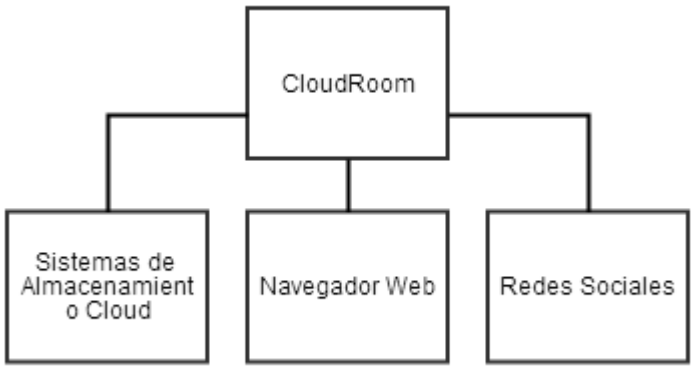

Figura 3.3 *Dependencias con sistemas externos*

2.2. Funciones del Sistema

El sistema deberá dar soporte a las siguientes tareas:

- Gestión de la Interfaz principal
- Interacciones sociales
- Gestión de cursos
- Acceso a cursos
- Acceso a Instituciones
- Acceso a perfiles de usuarios
- Mensajería
- Buscador

2.2.1. Gestión de la Interfaz principal

Para poder controlar desde la página de inicio toda la aplicación, será necesario definir las funciones principales del sistema, y unos accesos rápidos a las mismas, sin que se sature al usuario de información.

2.2.2. Interacciones sociales

Para dotar de funcionalidades sociales al sistema, este debe implementar una serie de funciones que permitan al usuario interaccionar con otras entidades del sistema de una forma similar a otras que ya le son familiares, debido a metáforas ya introducidas en otras redes sociales populares.

2.2.3. Gestión de cursos

Para poder ofrecer cursos y que los usuarios los sigan sin problemas, se debe permitir su creación y edición por parte de algunos usuarios, que actuarán a modo de docentes gestores de dichos cursos.

2.2.4. Acceso a cursos

Para seguir un curso se deberá proporcionar una interfaz y estilo a los usuarios, así como contenido incluido en los mismos. Para ello se introducirán mecanismos para el consumo de los usuarios.

2.2.5. Acceso a Instituciones

Con el objeto de proporcionar información al usuario, se deberá dar acceso a los perfiles oficiales de las Instituciones participantes, y que estas puedan publicar noticias, anuncios, relacionados con cursos o cualquier aspecto interesante para los usuarios.

2.2.6. Acceso a perfiles de usuarios

Para conseguir dar mayor funcionalidad social, y potenciar la colaboración entre usuarios, se debe permitir el acceso a perfiles personales de usuarios, ya sean alumnos o profesores, de forma que se puedan compartir archivos entre los mismos e incluso realizar grupos en los que colaborar en los cursos.

2.2.7. Mensajería

Para potenciar las comunicaciones sociales en el sistema, se deberá implementar un sistema principal de mensajería estilo email, así como también un sistema auxiliar de Chat para los usuarios que sean amigos entre sí.

2.2.8. Buscador

Para poder hacer búsquedas de cualquier entidad en la red social, se deberá implementar un buscador que esté siempre presente en la interfaz del usuario. Además se añadirán funcionalidades semánticas a la búsqueda.

2.3. Características de los Usuarios

El sistema se dirige a un público masivo, por lo que las características de uso del sistema deben seguir estándares y normas de usabilidad, con objeto de abrir su uso al mayor abanico posible de usuarios.

2.4. Restricciones

El acceso a la lista de cursos y su descripción general deberá ser público desde Internet, pero para poder cursarlos y realizar cualquier otra operación, se deberá ingresar en la plataforma previo registro de usuario.

2.5. Suposiciones y Dependencias

Se asume que estos requisitos son parciales y variables debido al carácter tan poco específico del sistema. Por lo que están sujetos a cambio.
El sistema *CloudRoom* sigue una Arquitectura Orientada a Servicios y será desplegado en un entorno Cloud, por lo que la disponibilidad del sistema está sujeta al número de usuarios y lo que se decida invertir en la infraestructura de AWS.

3. Requisitos Específicos

3.1. Requisitos funcionales

3.1.1. Gestión de la Interfaz principal

REQ1: Debe existir un acceso directo al perfil del usuario conectado, y que ese acceso directo muestre una retroalimentación de quién es el usuario actualmente conectado.

REQ2: Debe existir una barra principal horizontal, que filtre el contenido mostrado en función de la funcionalidad que se desee mostrar en cada momento. Esa barra contendrá al menos, los botones de: Todo, Cursos, Amigos, Instituciones, Noticias.

REQ3: La barra de navegación deberá mostrarse en todo momento, y contener una barra de búsqueda, el logo del sistema, y un acceso directo al perfil.

REQ4: Por defecto se mostrarán los cursos matriculados y las últimas noticias, un Chat lateral, y un menú de acceso al resto de funciones del sistema.

3.1.2. Interacciones sociales

REQ1: Un usuario debe poder crear nuevas conexiones sociales con otros usuarios. Al seguirlos, podrá visitar sus perfiles personales privados. Mientras no sean seguidores solo se tendrá acceso a una descripción general del usuario. Para seguir a un usuario, el usuario debe permitir previamente el acceso aceptando una petición de aquel que desee seguirle.

REQ2: Los usuarios deben poder enrolarse en cursos, añadirlos a lista cursos de interés si estos aún no han comenzado o deseen enrolarse en un futuro.

REQ3: Los usuarios pueden seguir a Instituciones, ya sean Organizaciones o Universidades.

REQ4: Los usuarios pueden ver publicaciones de aquellos de los que son amigos, conocidos o seguidores. Asimismo pueden indicar que una publicación de un usuario, o institución les ha gustado, recomendarla y así compartirla en sus perfiles, o comentar si el usuario que la ha publicado así lo permite.

REQ5: Las publicaciones pueden aceptar comentarios si el publicador lo desea. Para ello habrá una pestaña que aceptar comentarios.

REQ6: Al ver la información detallada de un curso, se mostrará una pequeña lista de hasta 5 cursos similares

3.1.3. Gestión de cursos

REQ1: Se debe permitir editar y modificar los cursos, sus elementos del panel así como también el contenido de los mismos. El contenido incluye: Estructura de la página web (menús, y su orden), creación de tests de evaluación, creación de tareas para evaluación y creación de correctores.

REQ2: Un curso solamente podrá ser creado por un usuario que posea el rol de Profesor.

REQ3: Los usuarios con rol de Profesor tendrán en su interfaz principal un botón especial de acceso al módulo de creación de cursos.

REQ4: El módulo de creación está formado por una plantilla de creación del curso, que contendrá: Título del curso, Institución que lo imparte, Descripción breve, Foto obligatoria para el curso, Vídeo divulgativo

del curso opcional, Carga de trabajo en horas/semana, Idioma en el que se imparte, Idiomas de Subtítulos disponibles, Periodo/s de impartición, Instructor/es, Acerca de con descripción detallada del trasfondo del curso, Programa detallado de temas del curso, Conocimientos previos recomendados o requeridos, Lecturas recomendadas, Formato de impartición (detalles sobre forma de examinar, uso de videos, técnicas de enseñanza utilizadas, etc.), Preguntas Frecuentes (FAQ).

REQ5: Una vez rellenada la plantilla se almacenará toda la información y se generará una nueva ventana con el aula virtual vacía. El profesor será el encargado de personalizar el contenido de los menús, así como crear contenido.

3.1.4. Acceso a cursos

REQ1: Se debe poder acceder públicamente a la descripción general de los cursos, para poder seguirlos o matricularlos. Dicha acción mandará a una pantalla de registro de usuario.

REQ2: Se debe poder acceder al contenido de aquellos cursos en los que un usuario esté enrolado: clases en video, foros, tareas, tests y materiales de estudio del curso.

3.1.5. Acceso a Instituciones

REQ1: Se debe poder consultar las páginas de las instituciones a las que un usuario siga. Dichas páginas serán públicas, pero al seguirlas, se notificará al usuario de cualquier novedad de las mismas desde su pantalla de inicio.

REQ2: El seguimiento de instituciones es libre, sin necesidad de autorización previa por parte de las mismas.

REQ3: Dentro de las páginas de las instituciones debe incluirse: Foto de portada, Logo, Descripción breve de hasta 400 caracteres de información básica de la institución, Lista de los cursos que ofrece, Lista de profesores impartiendo sus cursos en la plataforma, Localización, Dirección Web, URLs de redes sociales: Facebook, Twitter y Canal de YouTube. Además de un pequeño Feed de noticias y anuncios.

3.1.6. Acceso a Perfiles de Usuarios

REQ1: Cada usuario tiene un perfil público opcional donde se muestra información básica como: Foto de perfil público, Nombre y Apellidos, Rol (Alumno, Profesor, o Ambos), email (privado opcional), Descripción personal, Dirección Web Personal, Direcciones URL de redes sociales: Facebook, Twitter, LinkedIn, y de contacto.

REQ2: Cada usuario puede compartir publicaciones que sus seguidores podrán ver en sus feeds de la pantalla de inicio.

REQ3: Las publicaciones pueden tener privacidad personalizada.

REQ4: Cada usuario podrá compartir vía Dropbox carpetas con contenido a sus seguidores.

REQ5: Cada usuario podrá organizar un pequeño currículum en su perfil público.

3.1.7. Mensajería

REQ1: Deberá existir un servicio de mensajería privada entre usuarios similar a un servicio básico de correo.

REQ2: Deberá existir un servicio auxiliar de mensajería instantánea (Chat) entre los usuarios que sean seguidores.

3.1.8. Buscador

REQ1: Deberá existir una barra de búsqueda omnipresente en la interfaz, situada en la barra de navegación superior de la aplicación.

REQ2: Se deberá permitir la búsqueda de cursos, usuarios e instituciones.

REQ3: Se añadirá búsqueda por semántica, con determinada reglas de búsqueda.

3.2. *Requisitos no funcionales*

3.2.1. Requisitos de Interfaces Externas

3.2.1.1. Interfaces de Usuario

REQ1: La interfaz de usuario debe ser de tipo web y ejecutarse en un navegador.

REQ2: Debe ser multidispositivo, y ser adaptativa a la pantalla del usuario.

REQ3: Debe implementarse en HTML5, CSS3 y Javascript.

3.2.1.2. Interfaces Hardware

No se han definido.

3.2.1.3. Interfaces Software

3.2.1.3.1. Bases de datos

REQ1: El sistema debe almacenar los datos de todos los conceptos existentes en la plataforma, entre ellos los usuarios y sus interacciones sociales en una base de datos Neo4j.

REQ2: El sistema almacena los datos de las sesiones en la base de datos Clave-Valor, Redis.

REQ3: El sistema debe almacenar archivos importantes en el servicio Amazon S3. Incluye: Videos de las clases Apuntes

REQ4: El sistema almacenará en Amazon S3 el contenido dinámico de los cursos, publicaciones y otros en documentos JSON. Referenciados con URIs desde propiedades de Neo4j.

3.2.1.3.2. Sistema Operativo

REQ1: Se desplegará sobre un sistema operativo Linux, distribución Ubuntu 12.04.3 LTS.

3.2.2. Interfaces de Comunicación

REQ1: *La conexión al sistema y entre los diferentes servicios REST se hará exclusivamente a través de Internet.*

REQ2: *Las conexiones entre Servidores de bases de datos y los servicios de la aplicación que hagan uso de dichas bases de datos se harán a través de una red virtual definida por software, usando Amazon Virtual Private Cloud.*

3.2.3. Requisitos de Rendimiento

REQ1: *Para obtener un rendimiento aceptable, se desplegarán los servicios y las bases de datos en múltiples instancias, en modo de alta disponibilidad, y utilizando Amazon S3 y Cloudfront para distribución de contenido.*

3.2.4. Requisitos de Desarrollo

REQ1: *El ciclo de vida escogido será iterativo e incremental, dividiendo en módulos el sistema y desarrollándose en 3 ciclos, en cada uno de ellos añadiendo funcionalidad al sistema y refinando.*

REQ2: *Los lenguajes notacionales escogidos serán: UML para el código y la arquitectura software, Entidad-Relación extendido para los modelos de datos relacionales, y Redes Semánticas y jerarquía de Marcos para la base de datos de grafos.*

3.2.5. Requisitos de Seguridad

REQ1: *El sistema deberá garantizar la autenticidad de cada usuario, y controlar los privilegios de los mismos en función de los roles que tengan. Se deberá controlar que los usuarios solo puedan acceder a aquella información a la que están autorizados.*

REQ2: *Se deberá asegurar la confidencialidad e integridad de los datos de los usuarios.*

REQ3: *La autenticación deberá almacenar las contraseñas cifradas, y protegidas contra secuestros de la base de datos.*

REQ4: Las Interfaces REST del sistema deberán utilizar el protocolo oAuth.

3.2.6. Requisitos Tecnológicos

REQ1: Con el objetivo de que sea altamente disponible y accesible, el sistema se deberá desplegar en una plataforma Cloud, concretamente en Amazon Web Services.

3.2.7. Atributos

REQ1: Los usuarios se deberán dividir en roles que controlarán los privilegios que tienen cada uno de ellos. Estos roles son los de: Profesor y Alumno.

Capítulo 4
Solución Propuesta

En este apartado se describen los métodos, estrategias, tecnologías y decisiones de diseño escogidas para acometer el desarrollo del sistema planteado en el capítulo anterior.

4.1 Metodología y Ciclo de Vida

No se ha utilizado ninguna metodología específica de Ingeniería del Software sino que se han adoptado partes de algunas metodologías ágiles para este desarrollo concreto individual. Se han seguido una serie de técnicas ágiles como las historias de usuario, o la integración continua. Además, el ciclo de vida escogido ha sido iterativo e incremental (tal y como dictan este tipo de métodos de desarrollo software), desarrollando en ciclos cortos de aproximadamente un mes. Este tiempo ha variado debido al alto grado de experimentalidad del software que se estaba desarrollando.

4.2 Gestión de la Configuración Software

Se ha dado mucha importancia a este aspecto durante todo el desarrollo del proyecto. Los fallos en el software ocurren con más frecuencia de la que deberían, y los errores humanos también. Existen tecnologías que nos facilitan esta labor automatizando la gestión de librerías, de código y establecen mecanismos de gestión de versiones. En esta sección se comentaran las técnicas de Gestión de configuración utilizadas, así como las herramientas empleadas.

4.2.1 Integración Continua

Con el objetivo de que el código implementado diariamente pudiese ser ejecutado en la plataforma de despliegue final, se han utilizado técnicas y herramientas de integración continua. Además cada configuración realizada de una tecnología se ha ido documentando creando un *script* para automatizar el proceso de instalación, configuración y despliegue de bases de datos, entornos de ejecución y desarrollo.

4.2.1.1 Plataformas y Herramientas

En este apartado se describen aquellas herramientas y plataformas utilizadas para llevar a cabo las prácticas de integración continua mencionadas previamente.

Amazon EC2: Mediante la una conexión SSH a la instancia y un editor de código potente como *Emacs*, se ha desarrollado parte de la funcionalidad directamente en el entorno final de producción cuyo estado puede ser además almacenado en las imágenes de AWS denominadas *AMIs*. Permitiéndonos reproducir tantas veces como queramos ese entorno final de producción con todas las configuraciones necesarias.

Git: Gracias a esta herramienta se ha gestionado el control de versiones, integrando de forma muy frecuente el código desarrollado en diferentes ramas de funcionalidad. Permitiendo en cualquier momento integrar rápidamente en EC2 funcionalidades desarrolladas en una máquina local.

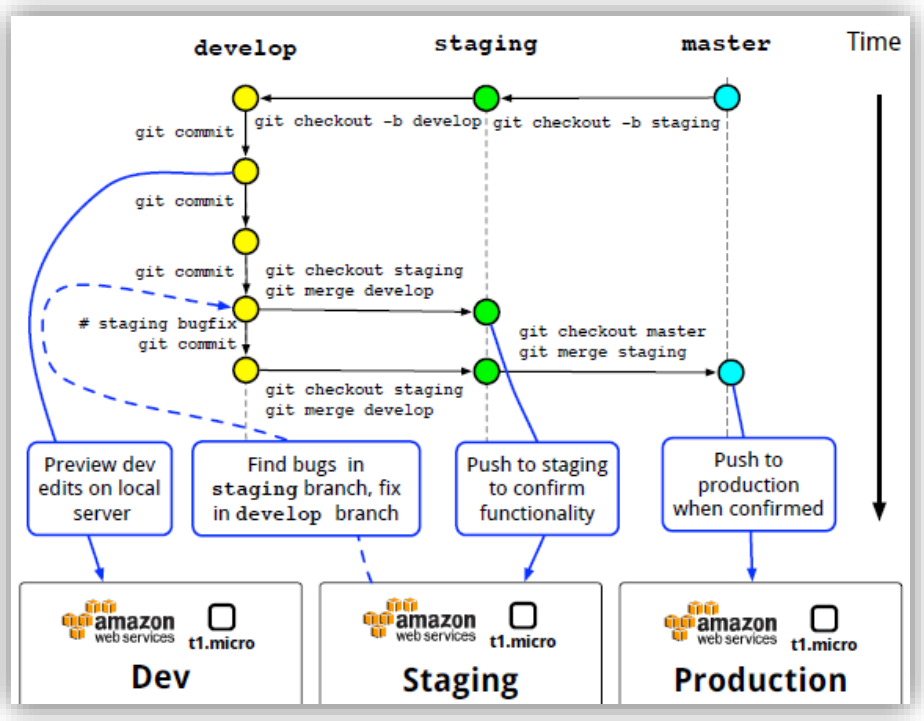

Figura 4.1 *Modelo de control de versiones del desarrollo en tres ramas*

Maven: Para automatizar la gestión de librerías Java, la construcción y el despliegue en el servidor Glassfish.

NPM (Node Package Manager): Este programa nos permite mediante el mantenimiento de un fichero *package.json*, gestionar un proyecto de *Node.js (*Nombre del proyecto, estructura del árbol de directorios del proyecto, sus librerías, repositorios asociados de control de versiones, entre otras funciones).

Scripts en Bash: Programación de Scripts para la automatización de la instalación y configuración de las siguientes tecnologías:

- Base de datos PostgreSQL
- Entorno de desarrollo Java JDK 7 y JavaEE 7 Web
- Base de datos Neo4j en Amazon EC2
- Base de datos Redis en Amazon EC2

4.2.3 Infraestructura y Entorno de Desarrollo

En esta subsección, se detallan tanto los sistemas operativos del entorno de producción y desarrollo, como todas aquellas librerías, herramientas, pilas de desarrollo utilizadas y sus correspondientes versiones.

Tipo de recurso	Nombre	Versión
Kit de Desarrollo	Oracle JDK	7u25-linux-x64
Librería (GlassFish 4)	Oracle Java EE SDK	7 Web-linux
IDE	Netbeans	7.3.1-linux
SGBD y Pila de desarrollo	PostgreSQL	9.2.4.1-linux-x64
Script de creación de la BD	users-dump.sql	0.1
AngularJS Framework	angular.zip	1.9
SGBD Neo4j Enterprise Startup	neo4j-enterprise-2.0.0-unix.tar.gz	2.00
SGBD Redis	redis-2.8.2.tar.gz	2.8.2
Repositorio de código	CloudRoom-SocialModule	No aplica
Repositorio de código	CloudRoom-Front-End	No aplica

4.2.4 Línea Base

Se han establecido los siguientes repositorios Cloud para la gestión de la línea base:

- Dropbox: Para la gestión de versiones documental.
- Bitbucket: Para la gestión de versiones del código.

4.3 Arquitectura del Software

En esta sección se especifica el diseño de alto nivel del sistema, y se argumentan las decisiones por las cuales se escogió finalmente. Se incluyen explicaciones y diagramas con el fin de documentar el trabajo realizado.

4.3.1 Escenario de Vista Lógica

El escenario (Figura 4.2) muestra la forma en que un usuario interactuará con el sistema. La vista es de muy alto nivel, y muestra 3 componentes completamente desacoplados entre sí.

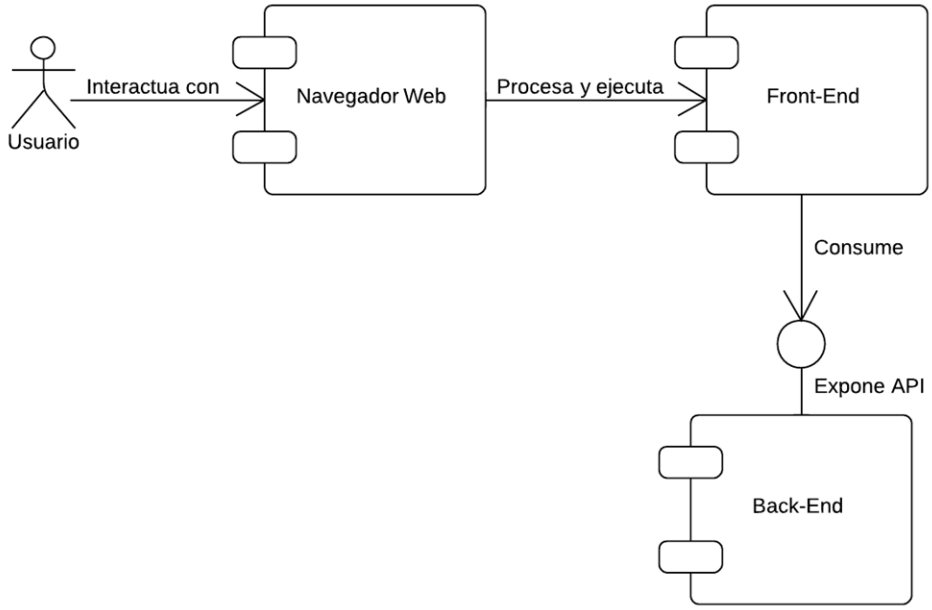

Figura 4.2 *Escenario de vista lógica del sistema*

El usuario ejecuta el navegador web en su máquina e interactúa con él. Es el navegador el que procesa la vista y ejecuta el Front-end alojado en la nube de AWS. El Back-end, alojado a su vez en AWS, recibe las peticiones del Front-end generadas por el usuario, y las procesa, realizando todas aquellas operaciones de lógica de negocio necesarias.

4.3.2 Arquitectura Lógica

Aquí se detalla el diseño de la arquitectura desde su vista lógica, mostrando aquellos componentes software que componen el sistema además de las interacciones entre los mismos.

CloudRoom se divide en dos capas principales: Front-end y Back-end. La primera de ellas incluye toda la funcionalidad relativa a la interfaz de usuario del sistema, tanto la vista como la lógica de la interfaz gráfica. La segunda acapara toda la lógica de negocio del sistema, aquellas operaciones costosas que el sistema realiza una vez que se ha dado orden desde el Front-end.
La separación entre Front-end y Back-end es total y las interacciones entre ambos se producen a consecuencia de las acciones del usuario en la interfaz. Si el Front-end necesita algo de lo que el Back-end es responsable, el primero envía una petición HTTP a la API REST del segundo, que responderá en un determinado momento con el recurso solicitado.

4.3.2.1 Vista de Caja Negra

El siguiente diagrama ilustra la vista lógica de caja negra del sistema, sin entrar en los detalles internos de cada componente.

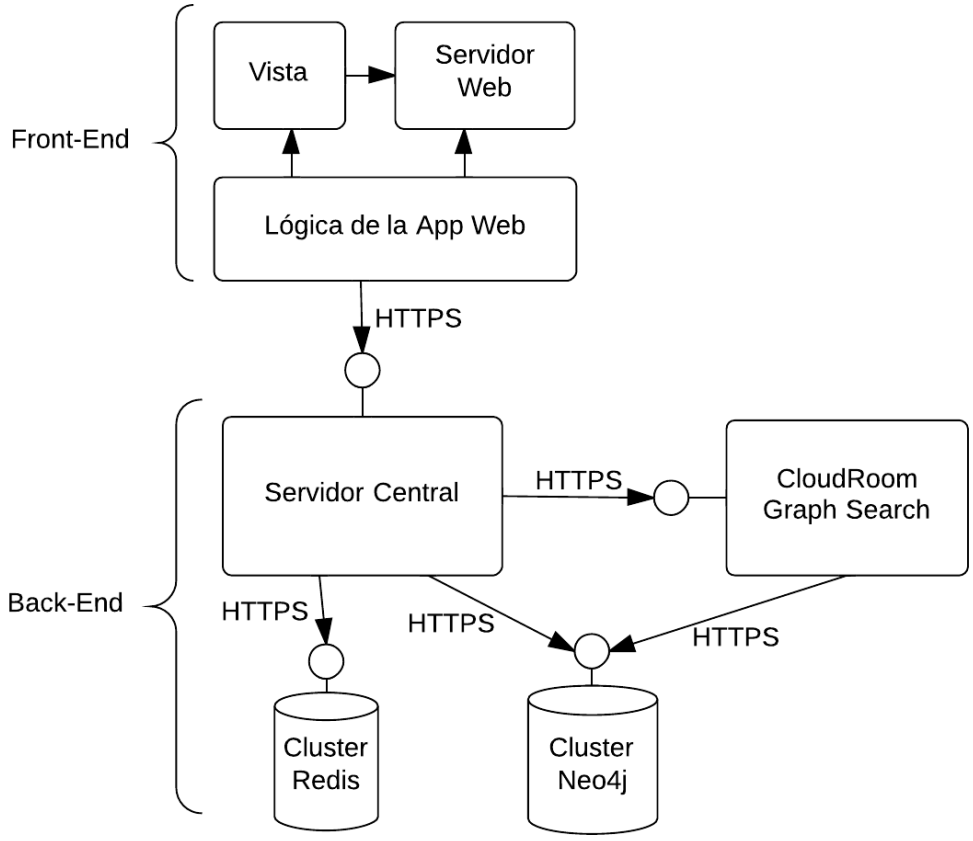

Figura 4.3 *Arquitectura lógica del sistema*

El Backend está desacoplado del Frontend. Esto se ha conseguido mediante el uso de una API REST. El servidor central ofrece una interfaz para que los clientes realicen operaciones contra el sistema. De esta forma, se pueden desarrollar fácilmente otros tipos de clientes que operen sobre el mismo Backend, como Android o iOS.
Las interacciones con las bases de datos también se producen mediante peticiones HTTPS a servidores REST que las propias bases de datos ofrecen. Se ha tomado esta decisión puesto que de esta forma se logran desacoplar todas las partes del sistema; permite balancear carga, tener servidores de respaldo, y escalar fácilmente. Además permite tener flexibilidad en el desarrollo de nuevas funcionalidades. A cambio, el sistema ve mermado ligeramente su rendimiento debido a las conexiones que se realizan.

Las interacciones entre cada módulo expuesto como servicio REST se producen mediante el protocolo HTTPS, lo que garantiza que las comunicaciones entre módulos van cifradas.

4.3.2.2 Vista de Caja Blanca

Una visión más en detalle (Figura 4.4) muestra los componentes que interactúan con el sistema.

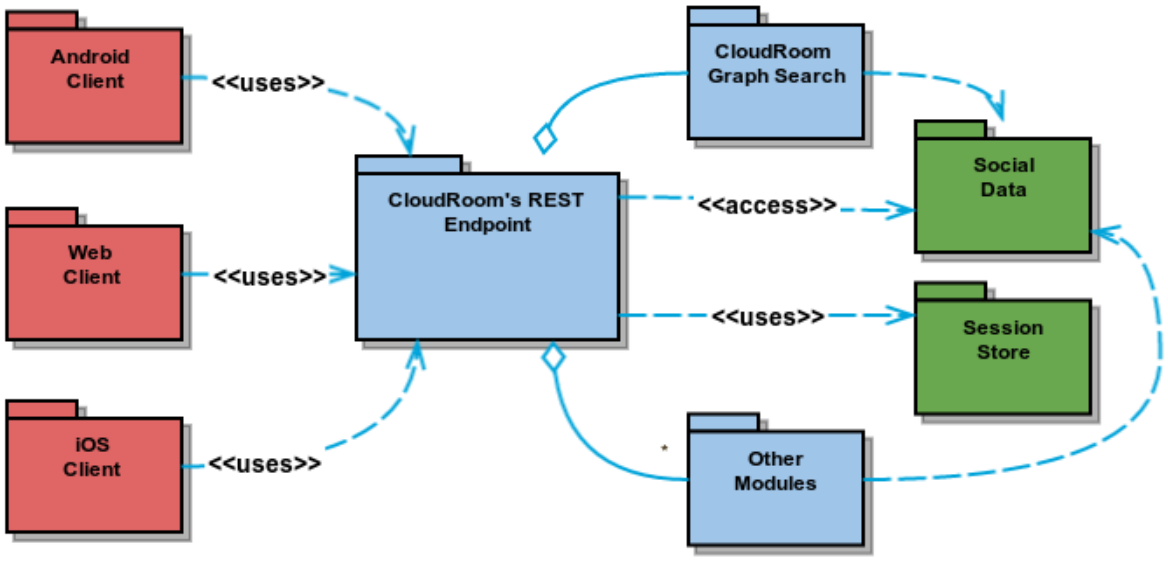

Figura 4.4 *Diagrama de paquetes con la vista de caja blanca del sistema*

En cuanto al Frontend, para este trabajo se ha diseñado la arquitectura del cliente web (Figura 4.5).

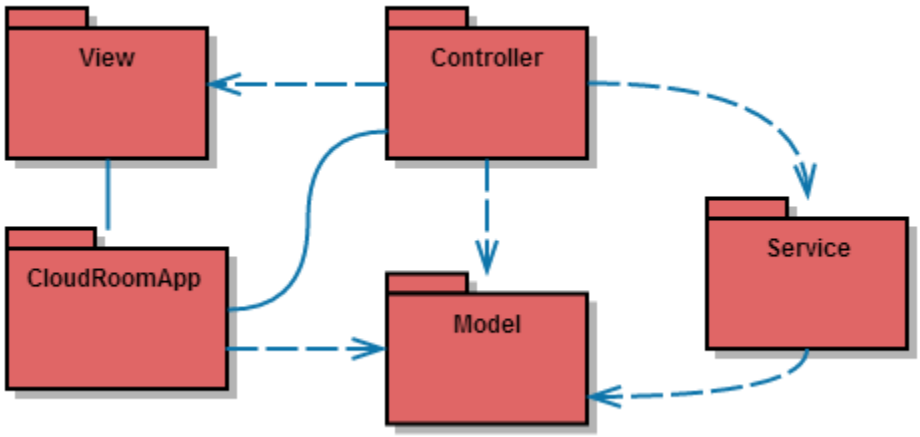

Figura 4.5 *Diagrama de paquetes con la vista de caja negra del sistema*

AngularJS nos ofrece una forma de estructurar nuestro Frontend mediante el patrón arquitectónico Modelo Vista Controlador (MVC), que nos da una solución elegante y nos permite tener un módulo altamente mantenible ante futuros desarrollos.
El diagrama de clases completo del sistema (Figura 4.6), modela el comportamiento relativo a la estructuración de los objetos del sistema. Se ha modelado mediante UML Color.

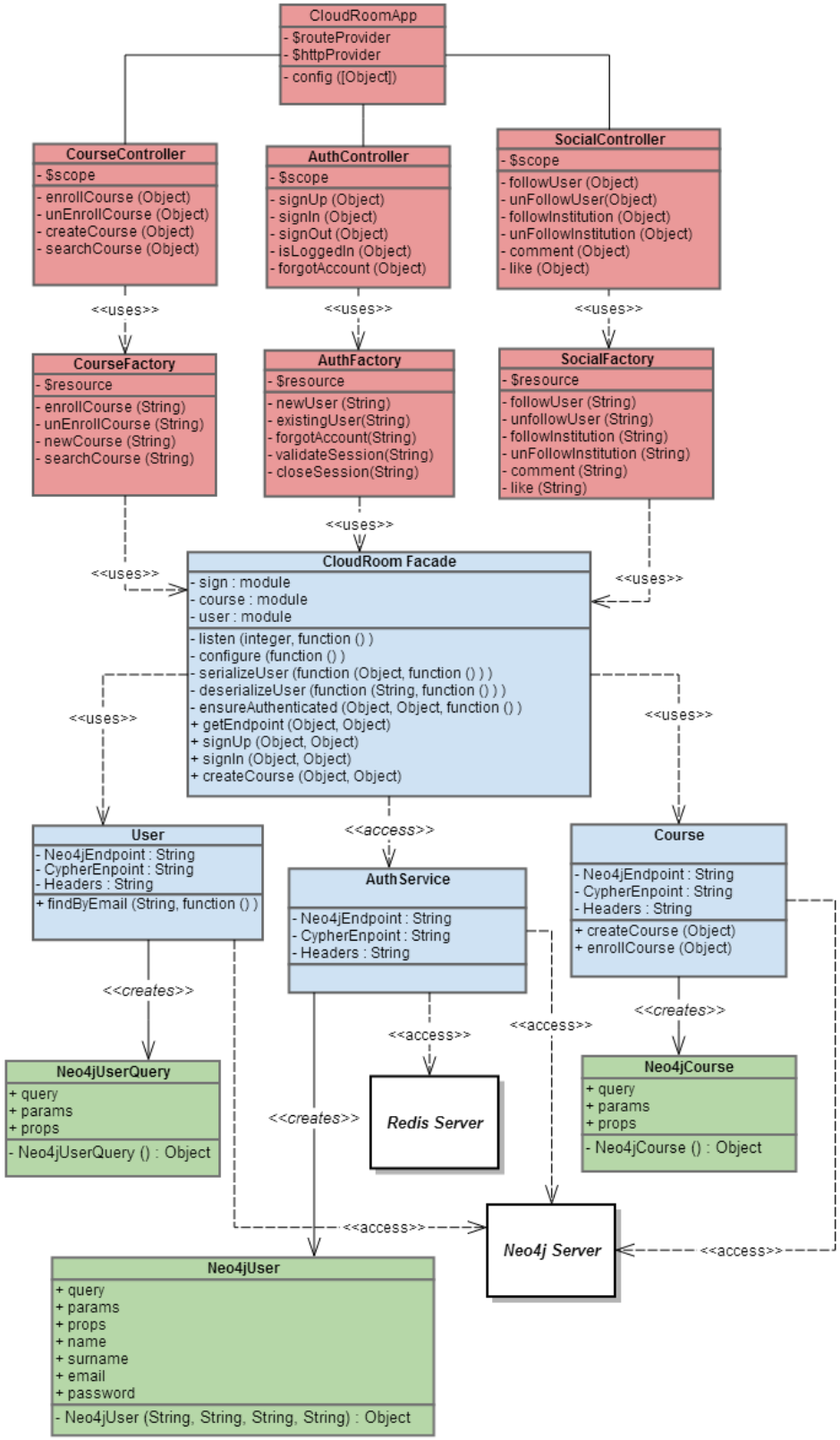

Figura 4.6 *Diagrama de objetos del sistema*

4.3.2.3 Modelo de Datos

Debido a que el problema al que nos enfrentamos es el diseño de un sistema que es fundamentalmente una red social, diseñar su modelo de datos supone un reto teniendo en cuenta el uso intensivo de los datos que hacen estos sistemas. Además, la potencial cantidad de usuarios del sistema, en un entorno distribuido y elástico requiere un diseño altamente mantenible, y lo más escalable posible.

En CloudRoom, el modelo de datos es el corazón del sistema. Se ha escogido una tecnología de base de datos muy novedosa y específica, que nos permite organizar los datos en un grafo social. *Neo4j* es una base de datos NoSQL de grafos.

El hecho de que este sistema gestor de base de datos maneje de forma nativa un grafo, nos proporciona de forma natural un modelo de datos que es rico semánticamente. A diferencia de utilizar cualquier otro tipo de base de datos, con esta tecnología, podemos de forma sencilla almacenar no solo datos, sino conceptos. Información consumible por humanos y por máquinas. Nos permite incorporar al sistema funcionalidades semánticas, propias de los sistemas que siguen las recomendaciones del W3C sobre Linked Data y Web Semántica: RDF, OWL, y SPARQL. Esto es así, puesto que el grafo de propiedades que presenta Neo4j es compatible con todas ellas. Un grafo de propiedades permite modelar tanto Hipergrafos, como Tripletas. Es posible por tanto almacenar Ontologías completas, e incluso realizar inferencias utilizando algoritmos de búsqueda en grafos, consultando el grafo mediante SPARQL o Cypher, o la propia API que ofrece Neo4j. Además se podría diseñar una capa semántica sobre dicho soporte de datos para ofrecer un motor de inferencias más completo.

Para diseñar el modelo de datos se han empleado técnicas de modelado con Redes Semánticas (Figura 4.7) y jerarquía de Marcos (Figura 4.8), puesto que para los propósitos de este proyecto, se ajustaba de forma natural al problema a resolver, sin embargo, puesto que Neo4j, nos permite gestionar las relaciones y las categorías de los nodos de forma distinta a como se haría en RDF u OWL, las relaciones modeladas son de un único sentido, pues Neo4j ya nos permite extraer mediante una API las relaciones entrantes y salientes de cada nodo.

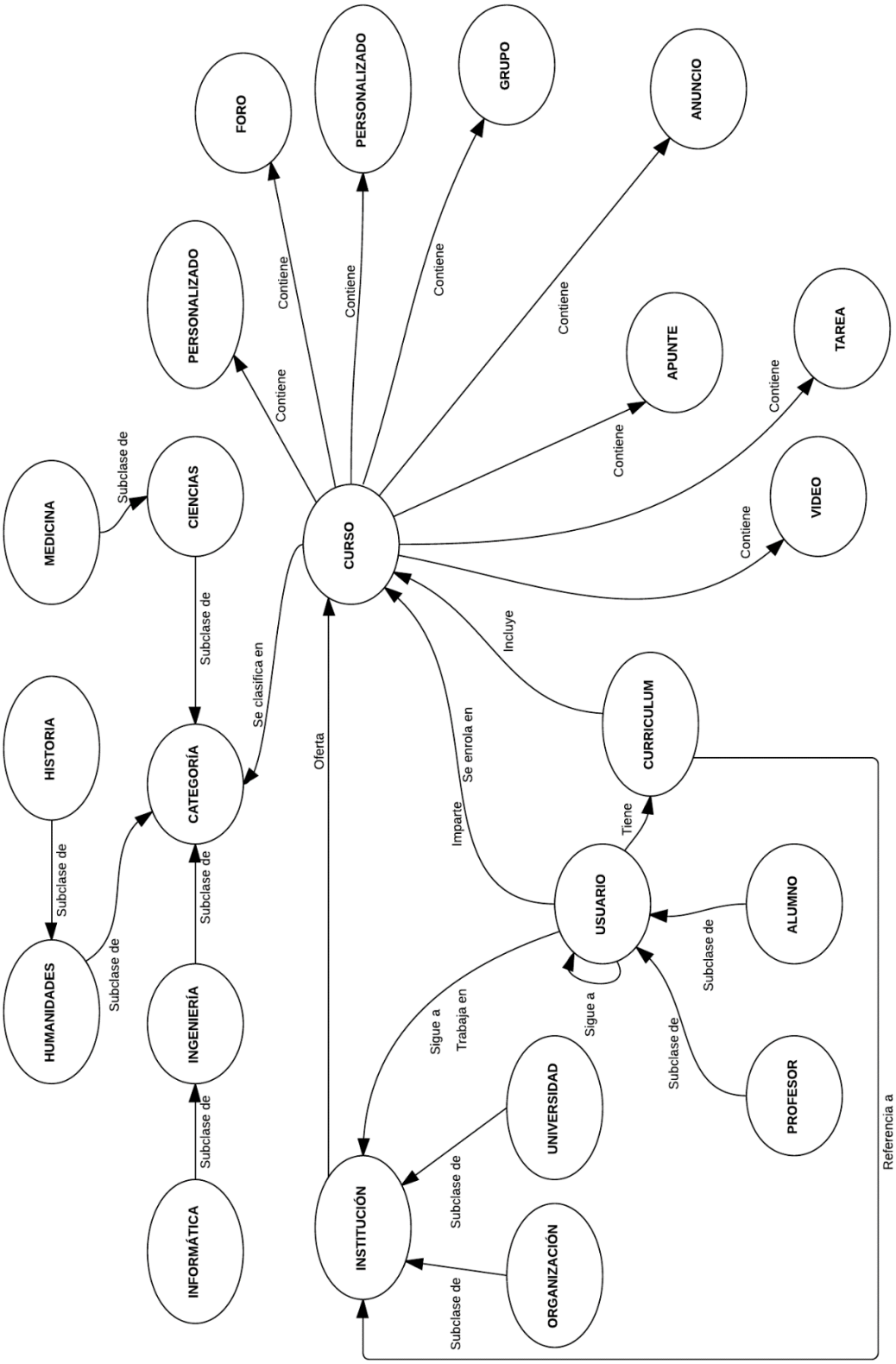

Figura 4.7 *Modelo de datos con redes semánticas*

En este primer diseño podemos ver el modelo de datos diseñado. Otra de las ventajas de usar una base de datos NoSQL, es que el esquema que introducimos es fácilmente mantenible y puede ir evolucionando con el tiempo de forma sencilla. En un proyecto de este tipo con una incertidumbre tal alta, esto es un punto fuerte.

Las relaciones del grafo permiten contener también propiedades, un aspecto importante para modelar históricos, o detalles de comportamiento de los usuarios.

Gracias a las nuevas funcionalidades introducidas en la versión 2.0 de Neo4j, podemos opcionalmente diseñar un esquema con ciertas restricciones para aquellos datos que se pueden introducir en la base de datos. Esto es muy interesante, pues para un sistema de este tipo, nos conviene introducir elementos que deben ser únicos (como puede ser el campo de email del usuario), y además nos permite añadir semántica a los datos. La semántica extra se consigue con la inclusión de los *Labels,* una funcionalidad que nos permite clasificar los nodos del grafo en categorías, las cuales además serán indexadas automáticamente por la base de datos para una consulta más rápida. Esas etiquetas además se asemejan a las clases padre de las jerarquías de marcos.

La jerarquía de marcos (Figura 4.8) muestra el esquema del dominio desarrollado para los propósitos de este TFG. Se ha modelado mediante UML.

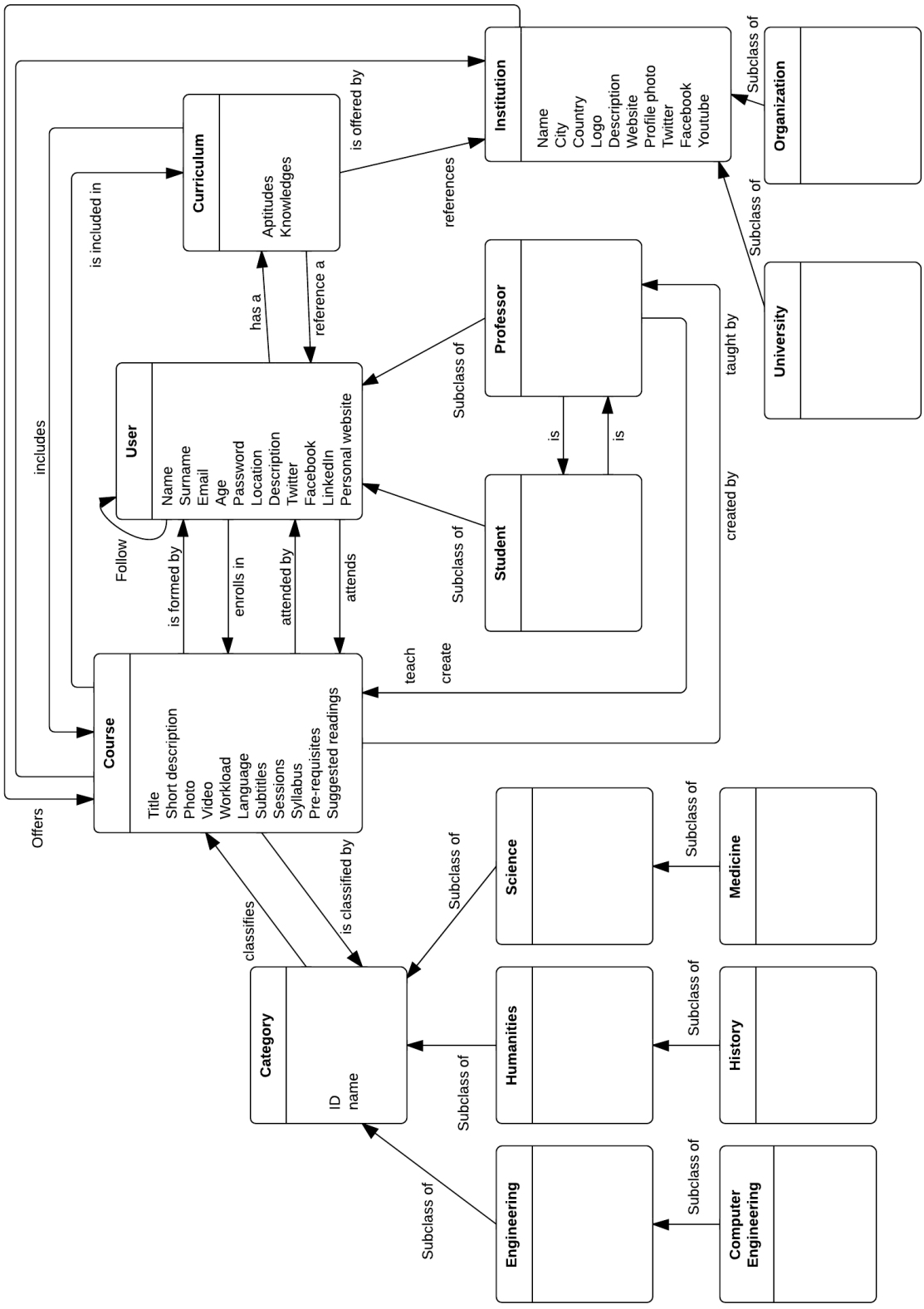

Figura 4.8 *Modelo de datos con Jerarquía de Marcos*

Una visión de grano más fino, nos sirve para definir el esquema final que Neo4j tendrá (Figura 4.9), utilizando la notación de nodos, relaciones, propiedades y etiquetas; propia de esta base de datos.

Se ha utilizado UML para detallarlo.

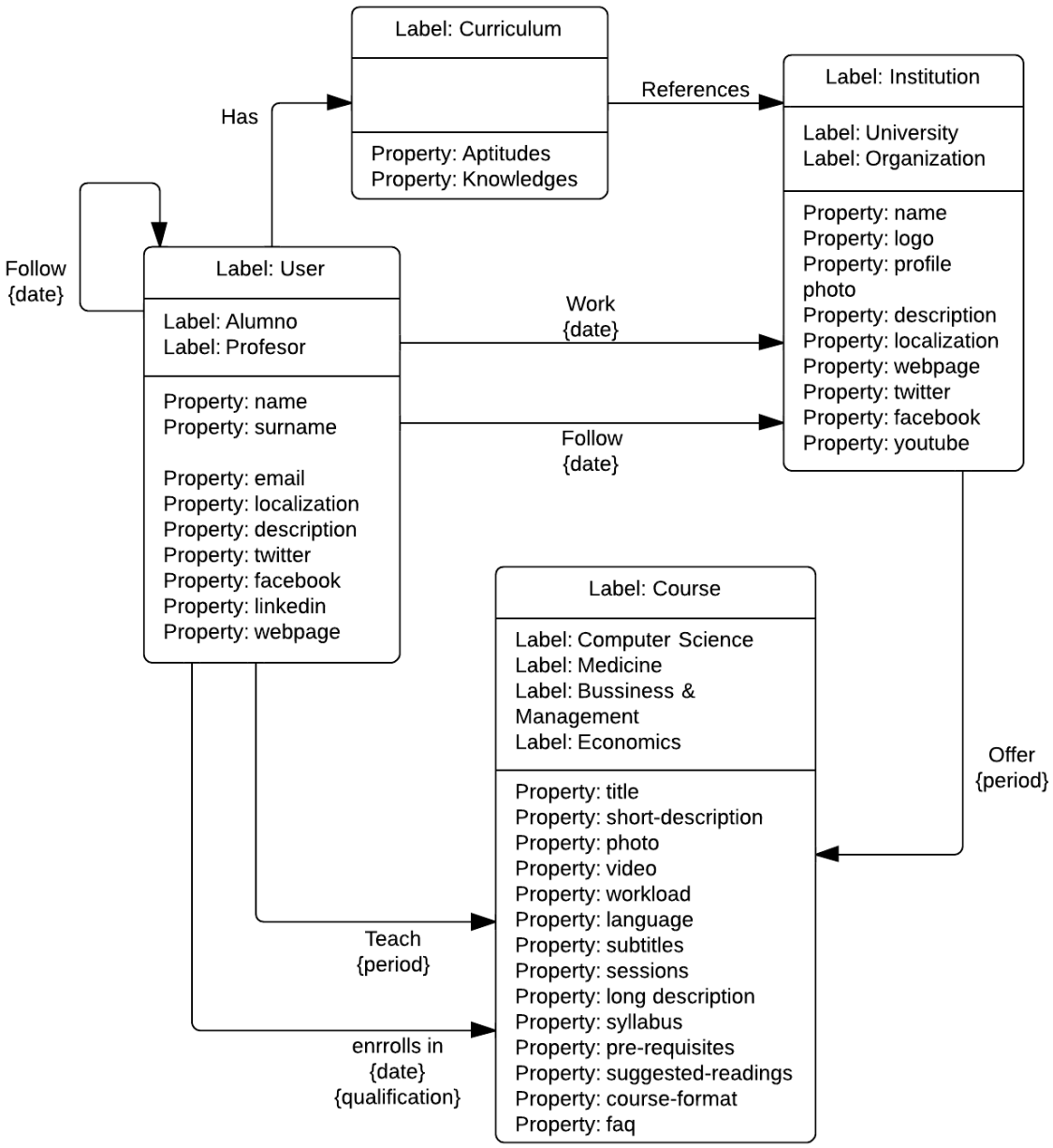

Figura 4.9 *Modelo de datos en Neo4j*

4.3.2.4 Diseño del Servicio REST

Para este trabajo, se ha diseñado una API REST para el Backend. Esta es la que da servicio a los clientes del sistema. Se han utilizado dos patrones de diseño de servicios REST. El primero de ellos es el *Data Transfer Object* (DTO), que utiliza un objeto como estructura serializable a JSON para su envió a través de una petición o respuesta.

El segundo, es el *Service Controller* que recibe aquellas peticiones al servicio en una unidad funcional, y delega operaciones de lógica de negocio a otras unidades funcionales. De esta manera, las operaciones contra el servicio son atómicas.

El modelo de recursos diseñado está formado por aquellas entidades principales almacenadas en la base de datos, en este caso, cada entidad es un nodo del grafo:

- Usuario
- Curso
- Institución

Cada recurso de los anteriores mencionados es una colección. Cada uno de los anteriores contiene numerosos subrecursos.
A continuación se detallan las operaciones principales.

Operación	**Registrar un usuario en el sistema**
Método	POST
Recurso	Controlador
URI	/SignUpUser
Cadena de Consulta	{ name, surname, email, password }
Devuelve	201 Created + (application/json)
	204 No Content

Operación	**Autenticar un usuario**
Método	POST
Recurso	Controlador
URI	/SignIn
Cadena de Consulta	{ email, password }
Devuelve	202 Accepted + (application/json)
	401 Unauthorized

Operación	**Validar una sesión de usuario**
Método	GET
Recurso	Controlador
URI	/ValidateSession
Cadena de Consulta	
Devuelve	202 Accepted + (application/json)
	401 Unauthorized

Operación	**Cerrar sesión de usuario**
Método	PUT
Recurso	Controlador
URI	/SignOut
Cadena de Consulta	
Devuelve	200 OK + (application/json)
	204 No Content

Operación	**Seguir a un usuario**
Método	POST
Recurso	Controlador
URI	/FollowUser
Cadena de Consulta	{ email }
Devuelve	200 OK + (application/json)
	204 No Content

Operación	**Dejar de seguir a un usuario**
Método	DELETE
Recurso	Controlador
URI	/UnFollowUser
Cadena de Consulta	{ email }
Devuelve	200 OK + (application/json)
	204 No Content

Operación	**Crear un curso**
Método	POST
Recurso	Controlador
URI	/CreateCourse
Cadena de Consulta	{ title, short-desc, institution, photo }
Devuelve	201 Created + (application/json)
	204 No Content

Operación	**Enrolarse en un curso**
Método	POST
Recurso	Controlador
URI	/EnrollCourse
Cadena de Consulta	{ title }
Devuelve	200 OK + (application/json)
	204 No Content

Operación	**Desenrolarse de un curso**
Método	DELETE
Recurso	Controlador
URI	/UnEnrollCourse
Cadena de Consulta	{ title }
Devuelve	200 OK + (application/json)
	204 No Content

4.4 Implementación, Verificación y Validación del Software

Esta sección recoge todos aquellos detalles sobre las decisiones de implementación del sistema. Desde los lenguajes de programación, de consulta o marcado, pasando por los Frameworks para esos lenguajes, así como las herramientas y tecnologías utilizadas durante el desarrollo. Para finalizar, se documentan las labores llevadas a cabo para la verificación de dicha implementación.

4.4.1 Lenguajes y Plataformas Utilizadas

Para construir un sistema complejo se requiere implementar muchas funcionalidades, de las cuales, muchas de ellas probablemente no se solucionarían de manera óptima con un mismo lenguaje. Además, cuando se diseña un sistema distribuido se debe tener muy en cuenta la escalabilidad del software que se quiere construir. La concurrencia en estos sistemas se hace indispensable, pues múltiples usuarios accederán de forma simultánea al sistema, y una solución bloqueante en este escenario, supone la negación del servicio.

En un principio, el lenguaje pensado para la implementación de la totalidad del sistema fue Java, se desarrolló un prototipo de un módulo de autenticación funcional. Sin embargo, a la hora de abordar el servidor central, se hizo patente que el servidor hacía un uso intensivo de la entrada/salida, y todo ello mediante el paso de JSONs a través de una API REST propia, y clientes REST a otros servicios como el de Neo4j. **Javascript** nos proporcionaba de manera nativa una forma de tratar JSON, y **Node.js** ofrecía una manera muy eficiente de implementar un servidor que hiciera un uso intenso de una base de datos para lectura y escritura.

La decisión de usar Javascript para el Frontend era ineludible, en cambio para la parte servidora se tomó una vez analizada la arquitectura del Backend.

CloudRoom, está estructurado en múltiples módulos mediante APIs RESTful, que utilizan JSON como formato de serialización. Mediante el uso de Javascript, se consigue casi a efectos prácticos, paso de objetos entre subsistemas independientes, lo que nos da una flexibilidad y productividad muy importante en el desarrollo, puesto que la des-serialización es trivial. Por todas estas razones, Node.js encaja a la perfección en nuestra aplicación.

Las consultas a la base de datos de usuarios se hacen mediante la API REST de Neo4j, y utilizando el lenguaje de consulta *Cypher* encapsulado en JSONs mediante Node.js. La base de datos Redis, se utiliza para almacenar las sesiones de los usuarios, y se accede a ella mediante la API del *middleware Connect-Redis* de Node.js.
La interfaz gráfica hace uso de HTML5 y CSS3, puesto que a pesar de no haber sido aprobados oficialmente aún por el W3C, son ya estándar *de facto* de la web. Su uso conjunto con el framework Bootstrap, nos proporciona de forma sencilla una interfaz con una alta usabilidad, con diseño *Responsive* y multidispositivo, con un coste muy reducido.

Por otro lado, se necesitaba diseñar un sistema, cuyas bases soportaran un desarrollo a larga escala. El Front-End requería operar potencialmente con múltiples servicios REST, y la productividad en el desarrollo era clave. AngularJS, nos proporciona de manera poco costosa funcionalidades que requerirían grandes cantidades de código en Javascript. Funcionalidades que además están siendo utilizadas por sistemas comerciales en producción, como es el caso de Google+, y otras aplicaciones de la compañía. Estas razones han bastado para escoger este framework, por encima de otros como Backbone.js o Knockout.js.

4.4.2 Verificación y Validación del software

Para probar el software que se ha implementado no se ha desarrollado un plan de pruebas, puesto que nos encontramos en una fase muy temprana del desarrollo, y no entraba dentro de los objetivos fundamentales del Trabajo. En su lugar, se han realizado pruebas unitarias de cada módulo desarrollado entendiendo módulo como una unidad funcional independiente. Además, se ha testeado siguiendo algunas pruebas de integración entre unidades funcionales. Estas pruebas se han hecho utilizando un estilo incremental, probando primero pequeñas partes de los subsistemas implicados en cada prueba. Un estilo no incremental produciría un caos a la hora de probar los diferentes componentes del software.

No se han desarrollado pruebas del sistema ni de aceptación puesto que el sistema no está desarrollado por completo, y además, el software es un desarrollo interno y no hay cliente al que hacerle pruebas de aceptación. Se ha seguido además un enfoque exclusivamente de caja negra, pues no se requería en estos momentos mayor nivel de cobertura de pruebas.

4.5 Despliegue en Amazon Web Services

En esta sección se describe todo lo relacionado con el despliegue en la plataforma Cloud IaaS de Amazon. Comenzando por la arquitectura física del despliegue de alto nivel, para después describir con un grano más fino en la arquitectura de bajo nivel. Se describe además en detalle los modos de alta disponibilidad utilizados para las bases de datos.

4.5.1 Arquitectura Cloud

Aparte del diseño detallado del sistema, y de su arquitectura física general para su despliegue en un sistema típico, se ha diseñado una arquitectura física para su despliegue en Cloud (Figura 4.10). Concretamente en la plataforma IaaS: Amazon Web Services, en adelante AWS. AWS nos da unas posibilidades mucho más amplias que cualquier PaaS. Para nuestro sistema, se necesita controlar muchos detalles del despliegue por lo que la decisión más adecuada era esta.

Debido a que la arquitectura software del sistema, no es trivial, se precisa gestionar a nivel de máquinas que participan en el despliegue, controlar aspectos de bajo nivel, como la alta disponibilidad de las bases de datos, cada una con una configuración distinta, algo que ningún PaaS nos permitiría. Sobra decir, que esto supone una mayor investigación de las tecnologías subyacentes, un diseño muy cuidado de la arquitectura, y como consecuencia un mayor tiempo en aspectos que están lejos de la implementación.

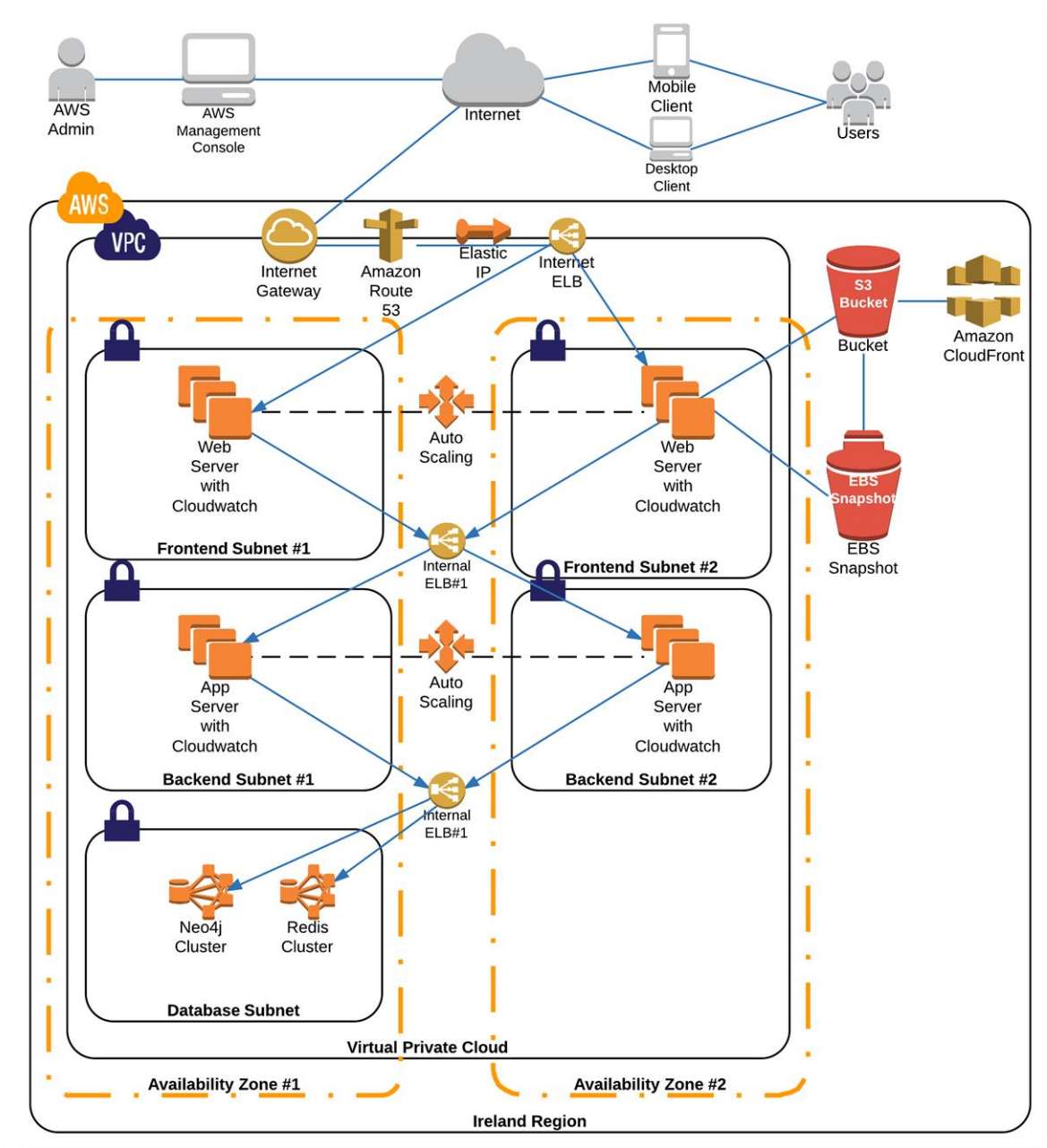

Figura 4.10 *Arquitectura Cloud de alto nivel en Amazon Web Services*

La figura muestra la arquitectura completa del despliegue en AWS. El sistema está alojado en la región de Irlanda, pues de todas las ofrecidas era la que mejor respondía a las necesidades de nuestro despliegue inicial; no se ha desplegado en más regiones, puesto que la complejidad que acarreaba no suponía ningún beneficio para los propósitos de este trabajo.

Como ya vimos en la sección 4.3, Arquitectura Software, la aplicación está estructurada en tres capas fundamentales: Capa de datos, Backend, y Frontend.

Se han agrupado las instancias AWS en una red virtual definida por software mediante el servicio *Virtual Private Cloud* (en adelante VPC), y dentro de esta red, para cada capa, se ha diseñado una subred dentro de la *VPC*, para agrupar lógicamente los distintos tipos de servidores, ya sean web, de aplicaciones, o de bases de datos. Se hace por motivos de seguridad y porque las bases de datos así lo requerían. Para el resto del sistema es simplemente una decisión organizativa. Se ha decidido utilizar dos zonas de disponibilidad para realizar auto-escalado y balanceo de carga. Se ha colocado un balanceador de carga para cada capa, debido a que el sistema está altamente desacoplado y se pretende que en un futuro la arquitectura se amplíe a nuevas capas.

Tanto el Frontend como el Backend disponen de dos instancias de respaldo en caso de la caída de una zona de disponibilidad. La capa de datos cuenta con varias instancias para implementar su propio modo de alta disponibilidad. Este despliegue responde ante situaciones de alta carga e incluso de caída de alguna instancia, sin embargo, supone un problema de garantía de servicio, puesto que si la zona de disponibilidad #1 fallara, el sistema quedaría sin servicio hasta que se recuperara dicha zona.

Los contenidos web y multimedia se almacenan en el servicio de S3 de AWS, de tal forma que se desacopla la distribución de contenido de las máquinas que dan servicio de aplicación. Además, se utiliza combinado con el servicio Cloudfront de distribución de contenido para mejorar la velocidad de distribución desde distintos puntos geográficos.

Cada instancia, dispone de dos volúmenes de datos (Figura 4.11), el *root* donde está el sistema operativo y sus recursos, y otro extra donde se instala el software desarrollado y se ejecuta cada módulo del sistema. Estos volúmenes son con los que se utiliza también el servicio S3, para guardar copias de seguridad de las instancias mediante *Snapshots*.

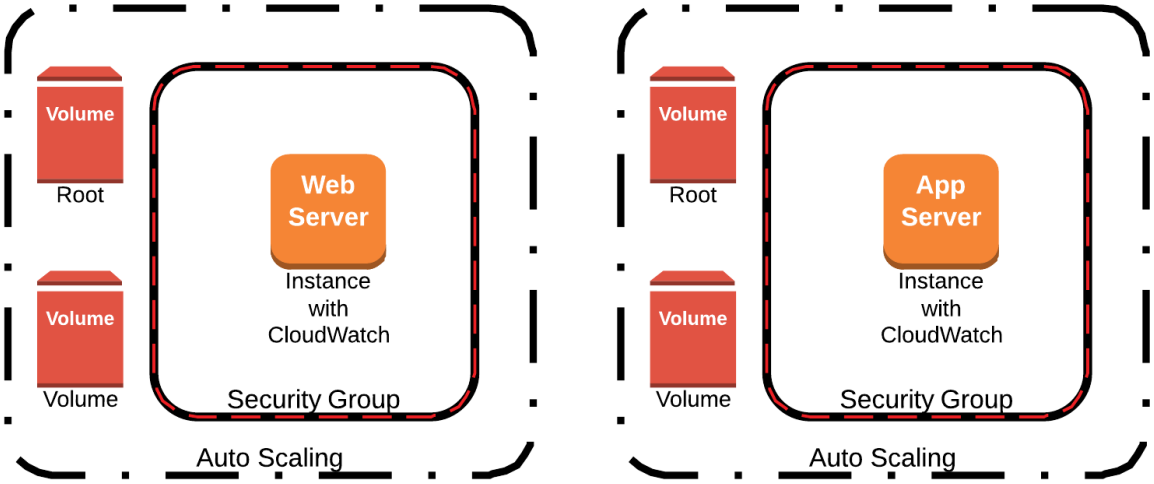

Figura 4.11 *Detalle del despliegue de los servidores web y de aplicaciones*

La capa de datos (Figura 4.12) utiliza diversas instancias para implementar dos modos Maestro-Esclavo distintos. El primero de ellos, el de *Neo4j*, presenta un esquema en el que el Maestro reparte carga entre los esclavos y además trabaja sirviendo peticiones. El segundo, el de *Redis* (aún en estado experimental), reparte incrementalmente los datos en tres nodos, y los esclavos de cada nodo actúan en caso de fallo del nodo maestro. El esquema de alta disponibilidad de *Neo4j*, se ha hecho así puesto que partir la información de un grafo es bastante complejo y en algunos casos imposible, en nuestro caso puede haber relaciones entre cada entidad del modelo de datos, por lo que partir la información por dominios de entidades es complicado.

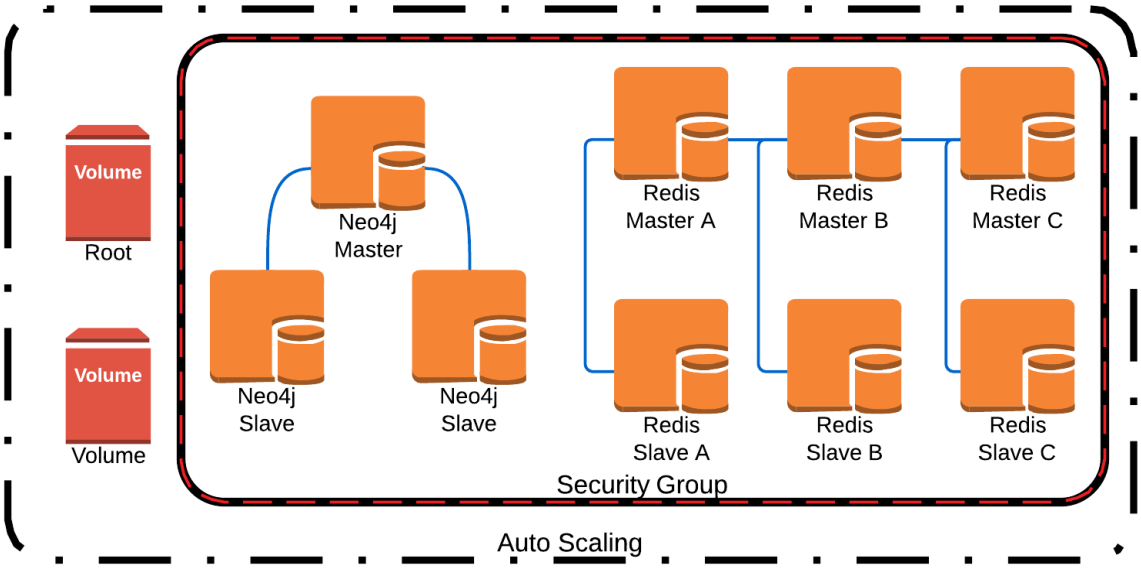

Figura 4.12 *Detalle del despliegue de los servidores de bases de datos*

Capítulo 5
Conclusiones

En definitiva, podemos afirmar que se ha realizado un trabajo integral de Ingeniería Informática. Es así, puesto que se han tenido que combinar prácticamente la totalidad de aquellas áreas de conocimiento fundamentales en Informática, así como varias ramas de estas. Comenzando por Bases de datos, Ingeniería del Software, Interacción Persona Ordenador, Inteligencia Artificial, Sistemas distribuidos, Sistemas Orientados a Servicios, Middleware para aplicaciones distribuidas, Matemática discreta (especialmente en lo relativo a teoría de grafos), Redes de Computadores, Programación y diseño de algoritmos, Concurrencia, Seguridad y Criptografía, Sistemas Operativos, Compiladores, llegando incluso a Arquitectura de Computadores. Sin duda, la combinación de los conocimientos de todas estas materias, ha sido clave para el desarrollo de este proyecto.

Por otro lado, el hecho de haber empezado un proyecto desde cero, y desarrollando una idea propia, ha marcado este proyecto de principio a fin. No tener unas pautas iniciales, el uso de tantas tecnologías y conceptos novedosos ha hecho que la implementación haya sido más lenta pero en cambio ha permitido trabajar otros conocimientos muy importantes a la hora de llevar un proyecto software. El análisis y especificación de requisitos, el diseño del sistema y su arquitectura, de su despliegue en alta disponibilidad, siempre pensando en una alta escalabilidad. En definitiva se ha ideado y diseñado las bases de un sistema distribuido robusto, altamente mantenible, capaz de responder ante millones de usuarios, y con las últimas tecnologías para este tipo de sistemas en la web. Se ha hecho una apuesta por utilizar tecnologías punteras y se ha demostrado que su orquestación era posible.

Se ha diseñado mediante numerosos servicios Cloud ofrecidos por Amazon Web Services, una arquitectura compleja, y se ha ahondado en aquellos aspectos que podrían ser necesarios para mantener un sistema de este tipo abierto a distintas zonas

geográficas del planeta, replicación de bases de datos, de los servidores, distribución elástica de los contenidos del sistema, etc.

Se ha aprendido a valorar distintas tecnologías y a escoger aquellas más adecuadas para un proyecto concreto, discerniendo sobre las ventajas y desventajas de: bases de datos, lenguajes de programación, diseños arquitecturales, así como decisiones que afectaran a la eficiencia del sistema.

Los meses que dura el desarrollo del trabajo dejan poco margen para indagar a fondo en todos los aspectos de un proyecto cuyas aspiraciones son enormes. No obstante, sirven como una magnifica guía tutorada para alguien que por primera vez va a realizar un proyecto de un tamaño ya considerable. Asimismo, dan un empuje fundamental si se quiere continuar el proyecto, tal y como este es el caso.

El Trabajo Fin de Grado, es en estos momentos una vía perfecta para orientar a los estudiantes, que en dicho instante están en su nivel de potencial más alto. No solo por la temática del mismo, sino también por la responsabilidad, y la oportunidad que le otorga al estudiante de demostrar todo aquello que ha aprendido durante sus estudios. Quizás sea la manera de despertar el ingenio, y hacer despegar la innovación que tanto nos falta.

Capítulo 6
Líneas Futuras

El hecho de haber enfocado este Trabajo con la idea de desarrollar un producto completo desde su concepción, haber diseñado pensando en un potencial uso masivo, comparándolo con otros sistemas similares en el mercado, hace que muchas de las investigaciones iniciales sobre funcionalidades a desarrollar se extiendan de manera asombrosa. La lista de funcionalidades recogida en las primeras fases del proyecto permitiría seguir con el proyecto durante meses.

6.1 Funcionalidades Educativas

Aquellas funcionalidades que dan auténtico valor a esta plataforma son aquellas que se centren en mejorar la experiencia de los estudiantes y que faciliten a los profesores la tarea de gestión de los cursos. Estas son algunas de las líneas de desarrollo que se pueden continuar en el futuro.

Las clases en video personalizadas son una de las señas de identidad de los MOOCs, a diferencia de las clases grabadas, estos videos se utilizan como un recurso educativo dirigido individualmente a los alumnos. Y no a un grupo como tal. Durante el proyecto se han definido las posibles soluciones para implementar esta funcionalidad.

Amazon S3 nos permite almacenar videos y servirlos bajo demanda y en alta disponibilidad gracias a CloudFront. Estos videos podrían estar estructurados en Neo4j como parte de un nodo curso, o como un nodo independiente etiquetado como Video que almacenara propiedades como su URL a S3 y algunos metadatos útiles para dotar de semántica a los datos sobre esas clases en video.

En un principio se pensó en usar MongoDB conjuntamente con GridFS para esto y la gestión de contenido de los cursos, pero Amazon S3, nos da un rendimiento superior combinándolo con el uso de una CDN y es menos costoso, pues nos proporciona la infraestructura de almacenamiento según la demanda que necesitemos en cada momento.

Al tener un carácter masivo y abierto al público, estos cursos no aspiran a ofrecer unas clases en las que los profesores puedan interactuar de manera individual con los alumnos. Las tareas deben estar completamente automatizadas para que este modelo masivo sea viable. Esto es uno de los desafíos a los que los MOOCs se enfrentan. Sin embargo, algunos de los sistemas que utilizan son viables y podrían ser implementados en el futuro. Los tests con corrección automática, las evaluaciones entre pares, los ejercicios que se ejecutan durante la reproducción de un video de clase y que pausan el video a modo de simular cuando un profesor pregunta a un alumno en clase.

Para corregir en gran medida los problemas de masificación de los cursos, una fórmula muy interesante es la de implementar funcionalidades colaborativas. Entre estas podemos implementar las de creación de grupos de trabajo privados entre alumnos, los foros para consulta de los alumnos, a través de los cuales los profesores pueden interactuar abiertamente con ellos. E incluso una nueva opción que Google nos proporciona, y es la del uso de Google Helpouts, un servicio informático al estilo de los bancos de tiempo que ofrecen consultas de casi cualquier cosa imaginable a especialistas. Estos especialistas pueden cobrar o no, y los alumnos disponen mediante un chat de video de un servicio online de clases particulares por parte de un experto.

6.2 Motor de Recomendaciones

Los motores de recomendaciones son tendencia en Internet, y gracias a Neo4j, implementar uno es relativamente sencillo. Mediante una arquitectura inteligente que siga el método de clasificación heurística guiada por los datos, podemos realizar un motor de recomendaciones de cursos para los usuarios, así como también otros muchos para diferentes problemas, como el de evaluar a estudiantes y recomendarlos para un determinado puesto de trabajo que una empresa requiera cubrir, ofreciendo así, un servicio de búsqueda de talento para las empresas.

6.3 Buscador Semántico

Cuando Facebook añadió entre sus funcionalidades su nuevo sistema de búsqueda *Graph Search* (Figura 6.1), puso el foco de atención en una tecnología que se está convirtiendo en algo habitual entre el usuario de a pie. Google *Knowledge Graph* o Twitter *Interest Graph*, son de hecho sistemas de búsqueda en grafo como lo es Facebook *Graph Search*. Neo4j nos permite de una forma más natural conseguir esto. Mediante el diseño de una gramática que interprete ciertas reglas de lenguaje natural, y genere dinámicamente consultas en Cypher, para poder realizar directamente en el buscador del sistema, búsquedas semánticas del tipo: "Cursos de mis amigos que viven en Madrid".

Figura 6.1 *Facebook Graph Search*

6.4 Chat Social

La implementación de un Chat para la plataforma que permite en grupos privados de trabajo realizar videoconferencias para estudiar o trabajar en equipo. Una de las posibles soluciones sería utilizar la API de Google Hangouts.

6.5 Certificados Firmados

Mediante la creación de un perfil de verificación de los usuarios, se puede ofrecer un servicio de entrega de certificados verificados y firmados por las Instituciones que hayan ofrecido el curso.

Bibliografía

[1] Balaji S. Srinivasan, 2013, Startup Engineering: Mobile, Stanford University, Course, [Online] Available at: <https://d396qusza40orc.cloudfront.net/startup%2Flecture_slides%2Flecture8-mobile-v2.pdf>

[2] "World Internet Users Statistics Usage and World Population Stats." 2002. 8 Oct. 2013 <http://www.internetworldstats.com/stats.htm>

[3] "Alexa Top 500 Global Sites." 2009. 8 Oct. 2013 <http://www.alexa.com/topsites>

[4] "What Is Web 2.0 – O'Reilly Media" 2005. 30 September. 2013 <http://oreilly.com/web2/archive/what-is-web-20.html>

[5] Key Facts - Facebook." 2012. 2 Nov. 2013 <http://newsroom.fb.com/Key-Facts>

[6] "GeoHive - Current World Population." 2012. 2 Nov. 2013 <http://www.geohive.com/earth/population_now.aspx>

[7] "IEEE MobileCloud 2014 | The 2nd IEEE International Conference on ..." 2008. 3 Nov. 2013 <http://www.mobile-cloud.net/>

[8] "Semantic Web roadmap." 3 Nov. 2013 <http://www.w3.org/DesignIssues/Semantic.html>

[9] "La Casa Blanca apoya el «cloud computing» - ABC.es." 2010. 1 Nov. 2013 <http://www.abc.es/20100705/tecnologia/casa-blanca-apoya-cloud-201007051403.html>

[10] "European Cloud Computing Strategy - Digital Agenda for Europe ..." 2012. 1 Nov. 2013 <http://ec.europa.eu/digital-agenda/en/european-cloud-computing-strategy>

[11] "AWS Case Study: NASA/JPL's Mars Curiosity Mission." 2012. 1 Nov. 2013 <http://aws.amazon.com/solutions/case-studies/nasa-jpl-curiosity/>

[12] "Massive Open Online Courses Are Multiplying at a Rapid Pace ..." 2012. 3 Nov. 2013 <http://www.nytimes.com/2012/11/04/education/edlife/massive-open-online-courses-are-multiplying-at-a-rapid-pace.html>

[13] "Daphne Koller - Technology as a Passport to Personalized ..." 2011. 3 Nov. 2013 <http://www.nytimes.com/2011/12/06/science/daphne-koller-technology-as-a-passport-to-personalized-education.html?pagewanted=all>

[14] "Coursera Wins "Best New Startup Of 2012," Get Schooled" 2013. 1 Jan. 2013 <http://techcrunch.com/2013/01/31/coursera-wins-best-new-startup-of-2012-get-schooled/>

[15] "John McCarthy - Wikipedia, la enciclopedia libre." 2005. 8 Oct. 2013 <http://es.wikipedia.org/wiki/John_McCarthy>

[16] Simson Garfinkel. "The Cloud Imperative | MIT Technology Review." 2012. 8 Oct. 2013 <http://www.technologyreview.com/news/425623/the-cloud-imperative/>

[17] "Who Coined 'Cloud Computing'? | MIT Technology Review." 2012. 1 Nov. 2013 <http://www.technologyreview.com/news/425970/who-coined-cloud-computing/>

[18] "Search Engine Strategies Conference - Google." 2006. 1 Nov. 2013 <http://www.google.com/press/podium/ses2006.html>

[19] "Amazon Web Services (España)" 2013. Jan. 2013 <http://aws.amazon.com/es/>

[20] "Above the Clouds: A Berkeley View of Cloud Computing" 2009. 10 Feb. 2013 <http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf>

[21] Martin Fowler. "NoSQL Distilled - Martin Fowler." 2012. 8 Oct. 2013 <http://martinfowler.com/books/nosql.html>

[22] "NoSQL Databases Explained | MongoDB." 2013. 8 Oct. 2013 <http://www.mongodb.com/nosql>

[23] Codd, Edgar F. "A relational model of data for large shared data banks." *Pioneers and Their Contributions to Software Engineering* (2001): 61-98.

[24] "Internet Connections and Big Data Graphs are Strategic Technology ..." 2013. 30 Oct. 2013 <http://www.neotechnology.com/gartners-top-10-strategic-technology-trends-internet-connections-and-big-data-graphs/>

[25] Ian Robinson, Jim Sebber and Emil Eifrem "Graph Databases" 2013. Book, O'Reilly. Available Online at: <http://graphdatabases.com/>

[26] "Sail Implementation · tinkerpop/blueprints Wiki · GitHub." 2011. 30 Oct. 2013 <https://github.com/tinkerpop/blueprints/wiki/Sail-Implementation>

[27] The Neo4 Team, "The Neo4j Manual v2.0.0" 2013. 8 Dec. 2013 <http://docs.neo4j.org/chunked/stable/index.html>

[28] Jonas Partner, Aleksa Vukotic, and Nicki Watt "Neo4j in Action" 2013. Book, Manning. Available Online at: <http://www.manning.com/partner/>

[29] "Peter Mika | Yahoo Labs." 2013. 2 Nov. 2013 <http://labs.yahoo.com/author/pmika/>

[30] Mika, Peter. "Ontologies are us: A unified model of social networks and semantics." *The Semantic Web–ISWC 2005* (2005): 522-536.

[31] "Linked Data | Linked Data - Connect Distributed Data across the Web." 2007. 2 Nov. 2013 <http://linkeddata.org/>

[32] "Big Data, Linked Data, Open Data | El Blog de Classora." 2012. 30 Oct. 2013 <http://blog.classora.com/2012/07/26/big-data-linked-data-open-data/>

[33] "Knowledge – Inside Search – Google." 2012. 30 Oct. 2013 <http://www.google.com/insidesearch/features/search/knowledge.html>

[34] "Freebase." 30 Oct. 2013 <http://www.freebase.com/>

[35] "Under the Hood: Building Graph Search Beta - Facebook." 2013. 30 Oct. 2013 <https://www.facebook.com/notes/facebook-engineering/under-the-hood-building-graph-search-beta/10151240856103920>

[36] "Introducing FlockDB | Twitter Blogs." 2013. 30 Oct. 2013 <https://blog.twitter.com/2010/introducing-flockdb>

[37] "LinkedIn's Data Infrastructure - InfoQ." 2010. 30 Oct. 2013 <http://www.infoq.com/news/2010/08/linkedin-data-infrastructure>

[38] "Bases de conocimiento en Internet | El Blog de Classora." 2012. 30 Oct. 2013 <http://blog.classora.com/2012/01/26/bases-de-conocimiento-en-internet/>

[39] Amit Singhal. "Introducing the Knowledge Graph: things, not strings - Google Blog." 2012. 30 Oct. 2013 <http://googleblog.blogspot.com/2012/05/introducing-knowledge-graph-things-not.html>

[40] "Why is mobile the future? - Quora." 2010. 7 Nov. 2013 <http://www.quora.com/Why-is-mobile-the-future>

[41] "iOS – Wikipedia The Free Encyclopedia" <http://en.wikipedia.org/wiki/IOS>

[42] "Android (operating system) – Wikipedia The Free Encyclopedia" <http://en.wikipedia.org/wiki/Android_(operating_system)>

[43] Douglas Crockford, "JavaScript: The World's Most Misunderstood Programming Language" <http://javascript.crockford.com/javascript.html>

[44] Balaji S. Srinivasan, 2013, Startup Engineering:, Stanford University, Course, Intermediate Javascript [Online] Available at:<https://d396qusza40orc.cloudfront.net/startup%2Flecture_slides%2Flecture10-intermediate-js.pdf>

[45] Kurt Cagle, "Turbo-charging JavaScript - Trace Trees and V8" <http://broadcast.oreilly.com/2008/09/turbo-charging-javascript---tr.html>

[46] Ryan Dahl, "Node's goal is to provide an easy way to build scalable network programs" <http://nodejs.org/about/>

[47] Express.js website <http://expressjs.com/>

[48] Restler - REST client library for node.js <https://github.com/danwrong/restler>

[49] Bcrypt-nodejs - Native implementation of bcrypt for NodeJS <https://github.com/shaneGirish/bcrypt-nodejs>

[50] Passport.js website <http://passportjs.org/>

[51] Redis session store for Connect <https://github.com/visionmedia/connect-redis>

[52] AngularJS website <http://angularjs.org/>

[53] Building Smartphone-Optimized Websites <https://developers.google.com/webmasters/smartphone-sites/details>

[54] HTML5 Introduction – W3schools.com <http://www.w3schools.com/html/html5_intro.asp>

[55] CSS3 Introduction – W3schools.com <http://www.w3schools.com/css/css3_intro.asp>

[56] Bootstrap from Twitter – Twitter Developers <https://dev.twitter.com/blog/bootstrap-twitter>

[57] <http://developers.slashdot.org/story/00/07/03/1855237/java-modeling-in-color-with-uml>

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
	Fecha/Hora	Mon Feb 17 10:36:16 CET 2014
	Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
	Numero de Serie	630
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)